Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут
імені Ігоря Сікорського”

Факультет електроніки
Кафедра звукотехніки та реєстрації інформації

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

КОНСПЕКТ ЛЕКЦІЙ

для студентів напрямів підготовки
6.050903 «Телекомунікації»,
6.050803 «Акустотехніка»
Спеціальностей
8.05090302 «Телекомунікаційні системи та мережі»
8.05080302 «Аудіо-, відео- та кінотехніка»

Рекомендовано Методичною радою факультету електроніки

Київ 2017

«Основи мікропроцесорної техніки»: Конспект лекцій з дисципліни «Основи мікропроцесорної техніки» / Уклад. Ю.О.Оникієнко, Д.В.Тітков: - Київ.: «КПІ імені І.Сікорського», 2017. - 109 с.

Гриф надано Методичною радою ФЕЛ
(Протокол № __9_ від 24.05.2018_ р.)

Навчальне видання

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

КОНСПЕКТ ЛЕКЦІЙ

Укладачі:

Оникієнко Юрій Олексійович, канд. техн. наук, ас.
Тітков Дмитро Валерійович, ас.

Відповідальний
Редактор

Рецензент

За редакцією укладачів

Зміст

Лекція 1. Програмно-апаратний комплекс Arduino ……. 26
Лекція 2. Мова С++. Програмування в Arduino ……35
Лекція 3. Цифрове введення / виведення ……. 38
Лекція 4. Асинхронний послідовний обмін …….43
Лекція 5. Архітектура мікропроцесорів …. 4
Лекція 6. Типова структура мікропроцесора ….. 9
Лекція 7. Система команд та структура мікроконтролера …. 17
Лекція 8. Функції часу …. 40
Лекція 9. Переривання …… 48
Лекція 10. Клас String …….. 52
Лекція 11. Двопровідний послідовний інтерфейс TWI (I^2C) …… 58
Лекція 12. Інтерфейс SPI на платах Ардуіно ……..66
Лекція 13. Інтерфейс 1-Wire ……72
Лекція 14. Основні напрями, завдання та алгоритми ЦОС ……. 78
Лекція 15. Подання цілих чисел в мікропроцесорних системах …. 83
Лекція 16. Організація обчислень в ЦПОС. Особливості архітектури …87
Лекція 17. Внутрішньокристальні емулятори. JTAG-емулятор … 97
Лекція 18. Програмування мікроконтролерів …102

Лекція 1. АРХІТЕКТУРА МІКРОПРОЦЕСОРІВ. Основні поняття.

1.1. Особливості організації процесу обробки інформації в цифрових пристроях (цифрових автоматах).

Завдання створення цифрового автомата, що виконує певні дії над двійковими сигналами, полягає у виборі елементів і способі їх з'єднання, що забезпечує задане функціональне перетворення. Ці завдання вирішують за допомогою математичної логіки або алгебри логіки.

Пристрої, що формують функції алгебри логіки, називають *логічними, або цифровими* і класифікують за різними відмітними ознаками. По схемному рішенню і характеру зв'язку між вхідними і вихідними змінними з урахуванням їх зміни по тактах роботи, розрізняють два типи цифрових пристроїв – *комбінаційні і послідовного типу*.

У комбінаційних цифрових пристроях сукупність сигналів на виходах в кожен конкретний момент часу повністю визначається вхідними сигналами, що діють у цей момент на його входах. Алгоритм функціонування комбінаційних пристроїв може бути представлений у вигляді таблиці відповідності, що містить значення вихідних сигналів для всіх можливих комбінацій значень вхідних сигналів.

Цифрові пристрої послідовного типу істотно відрізняються від комбінаційних, перш за все, наявністю пам'яті. Їх вихідні сигнали є функцією не тільки вхідних сигналів, але і внутрішнього стану, в якому пристрій знаходився до надходження вхідних сигналів.

На основі цифрового пристрою послідовного типу може бути спроектовано пристрій, який залежно від послідовності вхідних сигналів виконуватиме один з багатьох алгоритмів. Ці вхідні сигнали можуть розміщуватися і послідовно витягуватися із зовнішнього блоку регістрів, названого керуючою пам'яттю. Деякі вихідні сигнали можуть використовуватися для синхронізації надходження вхідних сигналів з пам'яті, що управляє, і для їх адресації. Такий пристрій може бути названий пристроєм з програмованою логікою, або *програмованим пристроєм*. До таких пристроїв відноситься і мікропроцесор.

Основні типи мікропроцесорних систем наступні:

мікроконтролери - найпростіший тип мікропроцесорних систем, у яких усі або більшість вузлів системи виконані у виді однієї мікросхеми;

контролери - керуючі мікропроцесорні системи, що виконані у виді окремих модулів;

мікрокомп'ютери - потужніші мікропроцесорні системи з розвинутими засобами сполучення з зовнішніми пристроями.

комп'ютери (у тому числі персональні) - самі потужні і найуніверсальніші мікропроцесорні системи.

Ми розглядаємо мікроконтролери. Мікроконтроллер має такі основні складові:

- ядро;
- пам’ять;
- периферія.

Ядро це, умовно кажучи, мікропроцесор. Почнемо з нього.

1.2 Архітектура мікропроцесорів (як частини МК)

Під архітектурою мікропроцесора мають на увазі складові частини мікропроцесора, а також їх взаємне з'єднання і взаємодію між ними. Архітектура мікропроцесора є логічна організація, що визначає можливості апаратної або програмної реалізації функцій, необхідних для побудови мікроЕОМ.

Архітектура включає:

1) структурну схему самого МП;
2) програмну модель МП (опис функцій регістрів);
3) інформацію про організацію пам'яті (місткість пам'яті і способи її адресації);
4) опис організації процедур вводу-виводу і управління;
5) опис системи команд і ін.

Існують два основних типи архітектури – фоннеймановська і гарвардська.

Фоннеймановську архітектуру запропонував в 1945 р. американський математик Джо фон Нейман. Особливістю цієї архітектури є те, що програма і дані знаходяться в загальній пам'яті, доступ до яких здійснюється по одній шині даних і команд. Прикладом такої архітектури є класичний мікропроцесор (I 8080).

Гарвардська архітектура вперше реалізована в 1944 р. в релейній обчислювальній машині Гарвардського університету (США). Особливістю цієї архітектури є те, що пам'ять даних і пам'ять програм розділені і мають окремі шину даних і шину команд, що дозволяє збільшити швидкодію МП системи за рахунок можливості одночасного звернення по цих двох шинах до пам'яті програм і пам'яті даних. Прикладом такої архітектури є мікроконтролер 8051 і всі представники сімейства MCS-51та інші сучасні МК.

Мікропроцесори визначаються наступними характеристиками:

- розрядність адреси і даних,
- тип корпусу,
- кількість джерел живлення,
- потужність розсіяння, температурний діапазон,
- можливість розширення розрядності,
- час циклу виконання команд (мікрокоманд),
- рівні сигналів,
- завадостійкість,
- здатність навантаження,
- об'єднання сигналів на виходах,
- надійність і т.ін.

По числу ВІС в комплекті (МПК) розрізняють *однокристальні, багатокристальні, багатокристальні секційні* мікропроцесори.

Однокристальні мікропроцесори утворюються при реалізації всіх апаратних засобів процесора в одній ВІС. По мірі збільшення ступеня інтеграції елементів в кристалі і числа виведень корпусу, параметри однокристальних мікропроцесорів поліпшуються. Проте можливості однокристальних мікропроцесорів обмежені апаратурними ресурсами кристала і корпусу. Тому поширеніші багатокристальні і багатокристальні секційні мікропроцесори.

Багатокристальні мікропроцесори виходять при розбитті його логічної структури на функціонально закінчені частини, які реалізують у вигляді ВІС. Функціональна закінченість ВІС багатокристального мікропроцесора означає, що його частини виконують наперед певні функції і можуть працювати автономно, а для побудови розвиненого процесора не вимагається організації великої кількості нових зв'язків і яких-небудь інших інтегральних схем (ІС).

Одним з можливих варіантів розбиття структури процесора є створення трьохкристального мікропроцесора, що містить ВІС операційного процесора, процесора, що управляє, і інтерфейсного процесора. Операційний процесор (ОП) служить для обробки даних, процесор, що управляє (УП), виконує функції вибірки, декодування і обчислення адрес операндів, а також генерує послідовності мікрокоманд. Автономність роботи і велика швидкодія ВІС дозволяють вибирати команди з пам'яті з більшою швидкістю, ніж ВІС ОП. При цьому в УП утворюється черга ще не виконаних команд, наперед готуються ті дані, які будуть потрібні ОП в наступних циклах роботи. Така випереджаюча вибірка команд економить час ОП на очікування операндів, необхідних для виконання команд програм. Інтерфейсний процесор (ІП) дозволяє підключити пам'ять і периферійні засоби до мікропроцесора. Велика інтегральна схема ІП виконує також функції каналу прямого доступу до пам'яті.

Вибрані з пам'яті команди розпізнаються і виконуються кожною частиною мікропроцесора автономно, і тому може бути забезпечений режим одночасної роботи всіх ВІС МП, тобто конвейєрний *потоковий режим виконання послідовності команд програми* (виконання послідовності з невеликим зміщенням у часі). Такий режим роботи значно підвищує продуктивність мікропроцесора.

Багатокристальні секційні мікропроцесори випускають у тому випадку, коли у вигляді ВІС реалізуються частини (секції) логічної структури процесора. Мікропроцесорна секція – це ВІС, призначена для обробки декількох розрядів даних або виконання певних керуючих операцій. Секційність ВІС МП визначає можливість «нарощування» розрядності оброблюваних даних або ускладнення пристроїв управління мікропроцесором при «паралельному» включенні більшого числа ВІС. Багатокристальні

секційні мікропроцесори мають розрядність від 2–4 до 8–16–32–64 біт і дозволяють створювати високопродуктивні процесори ЕОМ.

За призначенням розрізняють *універсальні* і *спеціалізовані* мікропроцесори. Універсальні мікропроцесори можна застосовувати для вирішення різноманітних завдань. Їх ефективна продуктивність мало залежить від проблемної специфіки вирішуваних задач. Спеціалізація МП, тобто його проблемна орієнтація на прискорене виконання певних функцій, дозволяє різко збільшити ефективну продуктивність при вирішенні тільки певних задач. Серед спеціалізованих мікропроцесорів можна виділити: мікроконтролери, орієнтовані на виконання складних послідовностей логічних операцій; математичні МП, призначені для підвищення продуктивності при виконанні арифметичних операцій за рахунок, наприклад, матричних методів їх виконання; МП для обробки даних в різних областях застосування і т.д. За допомогою спеціалізованих МП можна ефективно вирішувати складні задачі паралельної обробки даних.

По вигляду оброблюваних вхідних сигналів розрізняють *цифрові* і *аналогові* мікропроцесори. Самі мікропроцесори – це цифрові пристрої, проте вони можуть мати вбудовані аналого-цифрові і цифро-аналогові перетворювачі. Вхідні аналогові сигнали передаються в МП після перетворення в цифрову форму, обробляються, і після зворотного перетворення в аналогову форму, поступають на вихід. З погляду архітектури такі мікропроцесори є аналогові функціональні перетворювачі сигналів і називаються аналоговими мікропроцесорами. Вони можуть виконувати функції будь-якої аналогової схеми. Застосування аналогового мікропроцесора значно підвищує точність обробки аналогових сигналів, а їх відтворення розширює функціональні можливості за рахунок програмного налагодження цифрової частини мікропроцесора на різні алгоритми обробки сигналів.

Звичайно до складу однокристальних аналогових МП входять декілька каналів аналого-цифрового і цифро-аналогового перетворювачів. У аналоговому мікропроцесорі розрядність оброблюваних даних досягає 24 біт і більше, велике значення приділяється збільшенню швидкості виконання арифметичних операцій.

По характеру часової організації роботи розрізняють *синхронні* і *асинхронні* мікропроцесори. Синхронні мікропроцесори – це мікропроцесори, в яких початок і кінець виконання операцій задаються пристроєм управління (час виконання операцій в цьому випадку не залежить від виду виконуваних команд і величин операндів). Асинхронні мікропроцесори дозволяють початок кожної наступної операції визначити по сигналу фактичного закінчення виконання попередньої операції. Для ефективнішого використання кожного пристрою мікропроцесорної системи, до складу асихронно працюючих пристроїв, вводять електронні ланцюги, що забезпечують автономне функціонування пристроїв. Закінчивши роботу над якою-небудь операцією,

пристрій виробляє сигнал запиту, що означає його готовність до виконання наступної операції. При цьому функції природного розподільника робіт приймає на себе пам'ять, яка, відповідно до наперед встановленого пріоритету, виконує запити решти пристроїв по забезпеченню їх командною інформацією і даними.

По кількості виконуваних програм розрізняють *одно-* і *багатопрограмні* мікропроцесори.

У *однопрограмних* мікропроцесорах виконується тільки одна програма. Перехід до виконання іншої програми відбувається після завершення поточної програми.

У *багато-* або *мультипрограмних мікропроцесорах* одночасно виконуються декілька (звичайно декілька десятків) програм. Організація мультипрограмної роботи мікропроцесорних керуючих систем дозволяє здійснювати контроль за станом і управляти великим числом джерел або приймачів інформації.

Контрольні питання

1. Які основні типи цифрових пристроїв існують. Їх особливості.

2. Які різновиди архітектури мікропроцесорів існують та чим відрізняються.

3. Дайте перелік основних характеристик мікропроцесора.

4. Наведіть класифікацію мікропроцесорів за кількістю кристалів та за призначенням.

Лекція 2. Типова структура мікропроцесора

Типова структура мікропроцесора приведена на рис. 2.1. Мікропроцесор складається з трьох основних блоків: арифметико-логічного пристрою (АЛП), блоку внутрішніх регістрів і пристрою керування. Для передачі даних між цими блоками використовується внутрішня шина даних. *Арифметико-логічний пристрій* виконує одну з головних функцій мікропроцесора – обробку даних. Перелік функцій АЛП залежить від типу мікропроцесора. Деякі АЛП здатні виконувати безліч різних операцій, у інших набір операцій обмежений. Функції АЛП визначають архітектуру мікропроцесора в цілому. У більшості мікропроцесорів АЛП виконує наступні операції: складання, віднімання, І, АБО, виключаюче АБО, інверсію, зсув вправо, зсув вліво, інкремент позитивний і негативний.

Важлива складова частина мікропроцесора – *регістр*. Кожен регістр мікропроцесора можна використовувати для тимчасового зберігання одного слова даних. Деякі регістри мають спеціальне призначення, інші – багатоцільове. Останні називаються *регістрами загального призначення* (РЗП) і можуть використовуватися програмістом на його розсуд

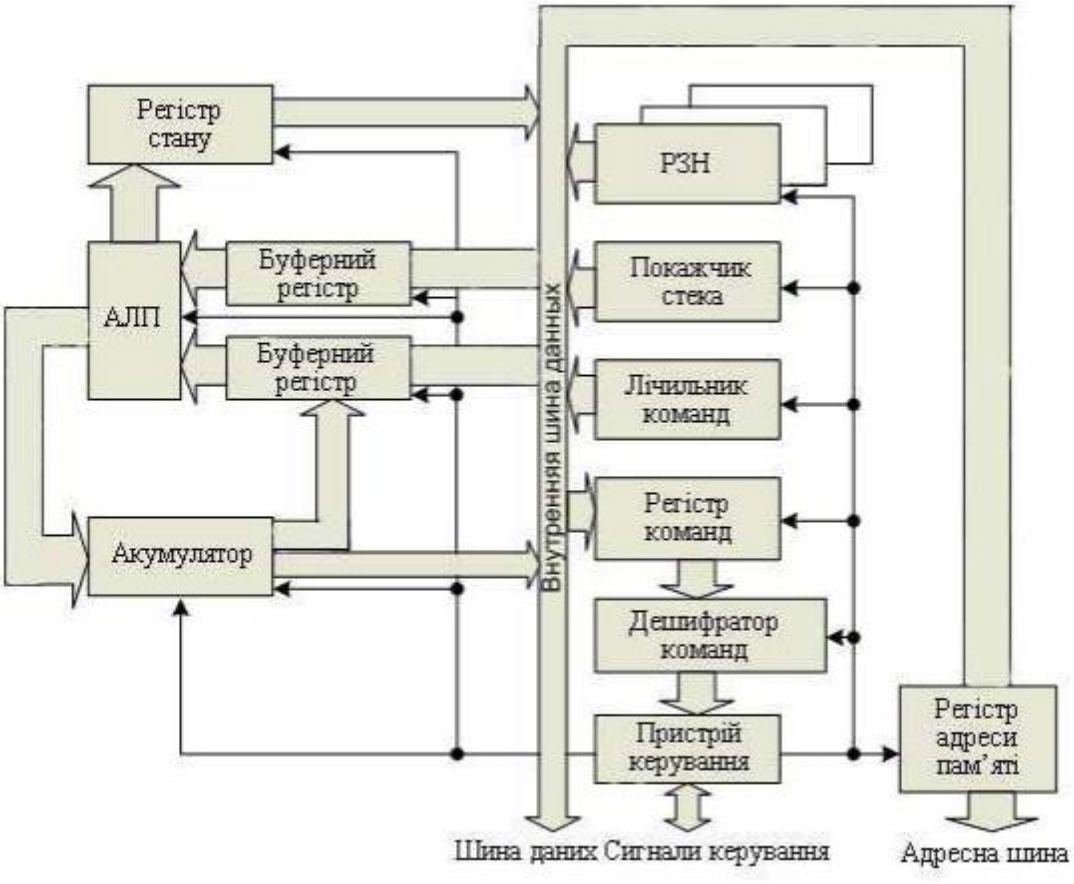


Рисунок 2.1 – Типова структурна схема мікропроцесора

Кількість і призначення регістрів в мікропроцесорі залежать від його архітектури.

Розглянемо призначення основних регістрів, наявних майже у всіх мікропроцесорах.

Акумулятор – це головний регістр мікропроцесора. Він використовується при різних маніпуляціях з даними. Більшість арифметичних і логічних операцій здійснюються шляхом використання АЛП і акумулятора. Будь-яка з таких операцій над двома словами даних (операндами) припускає розміщення одного з них в акумуляторі, а іншого – в пам'яті або якому-небудь регістрі. Так, при складанні двох слів, названих умовно А і В, і розташованих в акумуляторі та пам'яті відповідно, результуюча сума С завантажується в акумулятор, заміщаючи слово А. Результат виконання операції АЛП теж звичайно розміщується в акумуляторі, вміст якого при цьому втрачається.

Операцією іншого типу, що використовує акумулятор, є програмована передача даних з однієї частини мікропроцесора в іншу. Наприклад, пересилка даних між портом вводу – виводу і пам'яттю, між двома областями пам'яті і т.ін. Виконання операції «програмована передача даних» здійснюється в два етапи: спочатку виконується пересилка даних з джерела в акумулятор, потім – з акумулятора в пункт призначення.

Мікропроцесор може виконувати деякі дії над даними безпосередньо в акумуляторі. Наприклад, акумулятор можна очистити шляхом запису двійкових нулів у всі його розряди, встановити в одиничний стан шляхом запису у всі його розряди двійкових одиниць. Вміст акумулятора можна зрушувати вліво або управо, набувати його інвертованого значення, а також виконувати інші операції.

Акумулятор є найбільш універсальним регістром мікропроцесора. Для виконання будь-якої операції з даними, перш за все, необхідно помістити їх в акумулятор. Дані поступають в нього з внутрішньої шини даних мікропроцесора. У свою чергу, акумулятор може посилати дані на цю шину.

Кількість розрядів акумулятора відповідає довжині слова мікропроцесора, проте деякі мікропроцесори мають акумулятори подвійної довжини. У додаткові розряди акумулятора записуються при цьому біти, що з'являються при виконанні деяких арифметичних операцій. Наприклад, при множенні двох 8-бітових слів результат (16-бітове число) розміщується в акумуляторі подвійної довжини.

Лічильник команд – це один з найбільш важливих регістрів мікропроцесора. Як відомо, програма – це послідовність команд (інструкцій), що зберігаються в пам'яті мікроЕОМ, і призначених для того, щоб інструктувати машину, як вирішувати поставлену задачу. Для коректного її виконання команди повинні поступати в строго певному порядку. Лічильник команд забезпечує формування адреси чергової команди, записаної в пам'яті.

Коли мікропроцесор починає працювати, то по команді початкової установки в лічильник команд завантажуються дані з області пам'яті, заданої проектувальником мікропроцесора. Коли програма починає виконуватися, першим значенням вмісту лічильника команд є ця, наперед визначена, адреса.

На відміну від акумулятора, лічильник команд не може виконувати операції різного типу. Набір команд, що його використовують, украй обмежений в порівнянні з подібним набором для акумулятора.

Перед виконанням програми лічильник команд необхідно завантажити адресою, яка вказує на першу команду програми. Адреса першої команди програми посилається по адресній шині до схем управління пам'яттю, внаслідок чого прочитується її вміст за вказаною адресою. Далі ця команда передається в спеціальний регістр мікропроцесора, званий *регістром команд*.

Після витягання команди з пам'яті, мікропроцесор автоматично дає приріст вмісту лічильника команд. Цей приріст лічильник команд набуває в той момент, коли мікропроцесор починає виконувати команду, тільки що завантажену з пам'яті. Отже, з цієї миті лічильник команд містить адресу наступної команди.

Лічильник команд можна завантажити іншим вмістом при виконанні особливої групи команд. Може виникнути необхідність виконати частину програми, яка «випадає» з послідовності команд основної (головної) програми. Наприклад, таку частину програми, яка повторюється в процесі виконання всієї програми. Замість того щоб писати цю частину програми кожного разу, коли в ній виникає необхідність, програму записують один раз і повертаються до її повторного виконання, відступаючи від вказаної послідовності. Частина програми, що виконується шляхом відступу від послідовності команд головної програми, називається *підпрограмою*. В даному випадку в лічильник команд безпосередньо записується необхідна адреса. Часто лічильник команд має набагато більше розрядів, ніж довжина слова даних мікропроцесора. Так, у більшості 8-розрядних мікропроцесорів, число розрядів лічильника команд дорівнює 16.

Регістр команд містить команду в процесі її дешифровки і виконання. Вхідні дані поступають в регістр з пам'яті у міру послідовної вибірки команд. Звичайно існує можливість запису даних в регістр команд за допомогою набору перемикачів і кнопок на пульті управління ЕОМ. Як правило, цією можливістю користуються для передачі управління в початок програми.

Регістр адреси пам'яті при кожному зверненні до пам'яті мікроЕОМ указує адресу області пам'яті, що підлягає використанню мікропроцесором. Регістр адреси пам'яті містить двійкове число-адресу області пам'яті. Вихід

цього регістра називається *адресною шиною* і використовується для вибору області пам'яті або порту вводу-виводу.

Протягом вибірки команди з пам'яті, регістри адреси пам'яті і лічильника команд мають однаковий вміст, тобто регістр адреси пам'яті указує місцеположення команди, витягуваної з пам'яті.

Після декодування команди, лічильник команд одержує приріст на відміну від регістра адреси пам'яті.

В процесі виконання команди вміст регістра адреси пам'яті залежить від виконуваної команди. Якщо, відповідно до команди, мікропроцесор повинен провести ще одне звернення до пам'яті, то регістр адреси пам'яті підлягає вторинному використанню в процесі обробки цієї команди. Для деяких команд, наприклад команди очищення акумулятора, адресація до пам'яті не потрібна. При обробці таких команд регістр адреси пам'яті використовується лише один раз – протягом вибірки команди з пам'яті.

У більшості мікропроцесорів регістри адреси пам'яті і лічильника команд мають однакову кількість розрядів. Як і лічильник команд, регістр адреси пам'яті повинен мати в своєму розпорядженні кількість розрядів достатню для адресації будь-якої області пам'яті мікроЕОМ. У більшості 8-ми розрядних мікропроцесорів кількість розрядів регістра адреси пам'яті дорівнює 16.

Оскільки регістр адреси пам'яті підключений до внутрішньої шини даних мікропроцесора, він може завантажуватися від різних джерел. Більшість мікропроцесорів мають в своєму розпорядженні команди, що дозволяють завантажувати цей регістр вмістом лічильника команд, регістра загального призначення або якої-небудь області пам'яті. Деякі команди надають можливість змінювати вміст регістра адреси пам'яті шляхом виконання обчислень: нове значення вмісту цього регістра виходить шляхом складання або віднімання вмісту лічильника команд з числом, вказаним в самій команді. Адресація такого типу називається *адресацією з використанням зсуву*.

Буферний регістр – призначений для тимчасового зберігання (буферування) даних.

Регістр стану – призначений для зберігання результатів деяких перевірок, здійснюваних в процесі виконання програми. Розряди регістра стану приймають те або інше значення при виконанні операцій, що використовують АЛП і деякі регістри. Запам'ятовування результатів згаданих перевірок дозволяє використовувати програми, що містять переходи (порушення природної послідовності виконання команд).

За наявності в програмі переходу за заданою ознакою, виконання команд починається з деякої нової області пам'яті, тобто лічильник команд завантажується новим числом. У разі умовного переходу така дія має

місце, якщо результати певних перевірок співпадають з очікуваними значеннями. Вказані результати знаходяться в регістрі стану.

Регістр стану надає програмісту можливість організувати роботу мікропроцесора так, щоб за певних умов мінявся порядок виконання команд.

Розглянемо деякі найбільш часто використовувані розряди регістра стану.

1. *Перенесення/позика*. Даний розряд указує, що остання виконана операція супроводжувалася перенесенням або позикою (негативним перенесенням). Значення розряду перенесення встановлюється рівним 1, якщо в результаті складання двох чисел має місце перенесення із старшого розряду АЛП. Негативне перенесення (позика) фіксується в регістрі стану при відніманні більшого числа з меншого.

2. *Нульовий результат*. Приймає одиничне значення, якщо після закінчення операції у всіх розрядах регістра результату виявлені двійкові нулі. Установка цього розряду в 1 відбувається не тільки при негативному прирості вмісту регістра, але і при будь-якій іншій операції, результат якої – число з двійкових нулів.

3. *Знаковий*. Приймає одиничне значення, коли старший значущий біт вмісту регістра, куди записано результату операції, стає рівним 1. При виконанні арифметичних операцій з числами в додатковому коді одиничне значення старшого значущого біта показує, що в регістрі знаходиться негативне число.

Багато мікропроцесорів мають в своєму розпорядженні додаткові розряди станів. У деяких передбачені спеціальні команди для скидання або очищення всіх розрядів стану.

Регістри загального призначення (РЗП). Більшість МП мають в своєму складі набір регістрів, використовуваних як пристрої, що запам'ятовують. Оскільки АЛП може здійснювати операції з вмістом РЗП без виходу на зовнішню магістраль адрес і даних, то вони відбуваються набагато швидше, ніж операції із зовнішньою пам'яттю. Тому іноді РЗП називають надоперативною пам'яттю. Кількість РЗП і можливості програмного доступу до них у різних мікропроцесорів різні.

Покажчик стека. Стек – це набір регістрів мікропроцесора або елементів оперативної пам'яті, звідки дані або адреси вибираються «зверху» за принципом: перший – що поступив останнім.

При записі в стек чергового слова всі раніше записані слова зміщуються на один регістр вниз. При вибірці слова із стека, слова, що залишилися, переміщаються вгору на один регістр. Вказані процедури ілюструє рис. 3.2 Тут стек складається з семи регістрів. Якщо в стек завантажується яке-небудь слово, наприклад А5, то воно записується у верхньому регістрі, а кожне із слів А1...А4 переміщається в сусідні нижні регістри. Якщо ж А5 витягується із

стека, то кожне із слів A1..A4 переміщається в сусідні верхні регістри. Не можна витягнути A4 раніше A5, тобто автоматично реалізується відмічений вище принцип. Стек звичайно використовується в мікропроцесорах для зберігання адрес повернення при зверненні до підпрограм, а також для запам'ятовування стану внутрішніх регістрів при обробці переривань.

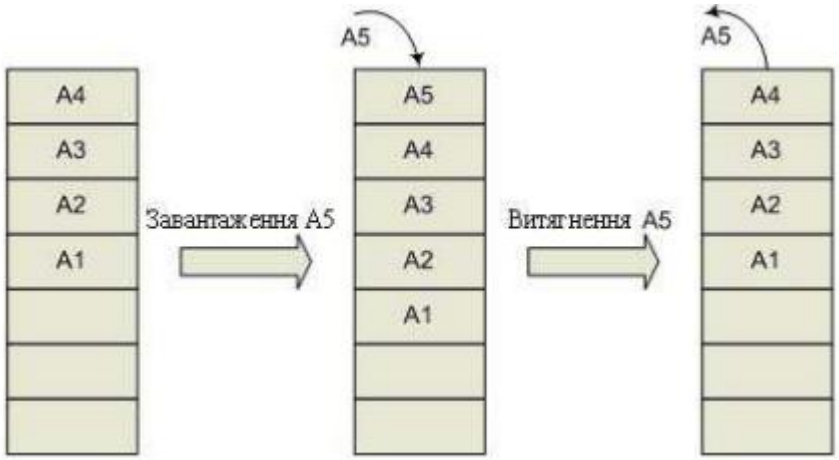


Рисунок 3.2 – Процедури роботи стека

При організації стека в пам'яті, час на звернення до нього буде дорівнювати циклу звернення до пам'яті. Ця операція виконується значно швидше, якщо стек у вигляді набору регістрів входить до складу мікропроцесора. Важливим параметром у такому разі є число регістрів стека. При спробі записати в стек більшої кількості слів, чим число його регістрів, перше слово буде втрачено. У деяких мікропроцесорах при переповнюванні регістрів стека відповідні слова записуються в стек пам'яті.

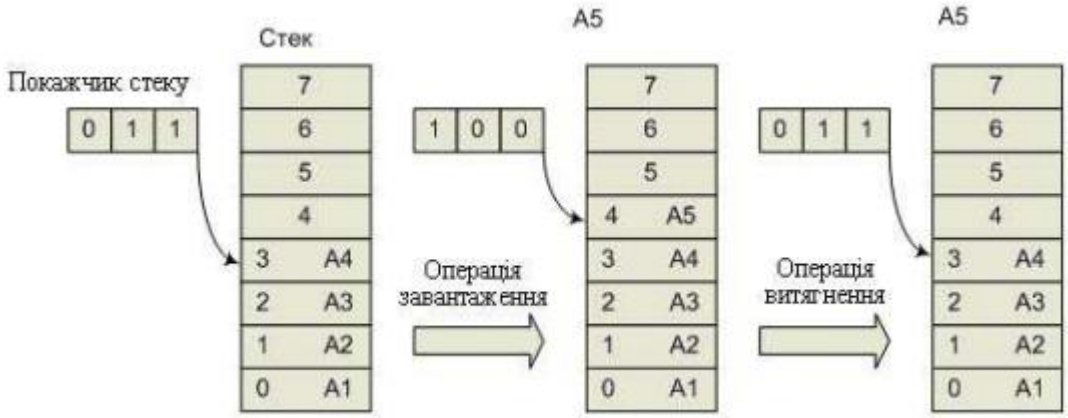


Рисунок 3.3 – Адресація елементу стека з використанням покажчика стека

Часто стек реалізується таким чином, що процес його функціонування нагадує роботу з пачкою документів, коли кожен новий документ кладеться зверху пачки. При такій організації стека необхідний *спеціальний регістр – покажчик стека* (РПС) для зберігання адреси останнього, за часом надходження, елементу стека. Приведений на рис. 3.3.

покажчик стека є трьохрозрядний регістр, з двійковим представленням інформації. Спочатку покажчик стека містить число 011_2. Це означає, що останній елемент – «верхівка стека» знаходиться в регістрі з адресою 011_2 (або 3_{10}). При операції завантаження в регістр 4_{10} записується число А5, а вміст покажчика стека змінюється так, що він указує на регістр 4_{10}. При операції витягання із стека проводяться зворотні дії.

Пристрій керування. Роль пристрою керування в мікропроцесорі полягає в підтримці необхідної послідовності функціонування всієї решти його ланок.

По сигналах пристрою керування чергова команда витягується з регістра команд. При цьому визначається, що необхідно робити з даними, а потім забезпечується послідовність дій для виконання поставленого завдання.

Одна з головних функцій пристрою керування – декодування команди, що знаходиться в регістрі команд, за допомогою дешифратора команд, який в результаті видає сигнали, необхідні для її виконання.

Крім вказаних вище дій, пристрій керування виконує деякі спеціальні функції керування послідовністю включення живлення і процесами переривань. Переривання – це свого роду запит, що поступає на схеми управління з інших пристроїв (пам'яті, вводу-виводу). Переривання пов'язане з використанням внутрішньої шини даних мікропроцесора. Пристрій керування приймає рішення, коли і в якій послідовності інші пристрої можуть користуватися внутрішньою шиною даних.

Система шин. На характеристики мікропроцесора впливає спосіб організації його зв'язку із зовнішнім середовищем – пристроями вводу-виводу (ПВВ) і пристроями, що запам'ятовують. За способом організації зв'язків із зовнішнім середовищем розрізняють мікропроцесори з мультиплексованою шиною адреси і даних (рис. 3.4, а) і з роздільними шинами адреси і даних (рис. 3.4,б) [12]. Мікропроцесор з роздільними шинами адрес і даних зображений на рис. 1.1.

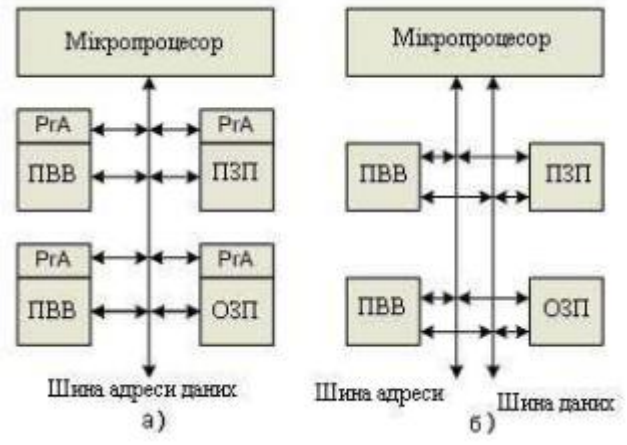


Рисунок 3.4 – Система шин мікроЕОМ

У мікропроцесорах з мультиплексованою шиною адреса зберігається на шині тільки короткий проміжок часу, тому пристроям, підключеним до шини, потрібні регістри адреси (РГА). Для організації обміну інформацією в таких мікропроцесорах необхідно використовувати керуючий сигнал «дані-адреси». При роздільних шинах адреси і даних сигнал, що управляє, не потрібен. Крім того, у пристроїв, підключених до шин, відпадає необхідність в регістрі адреси, оскільки він може бути розміщений безпосередньо на кристалі мікропроцесора. Розрядність адресної шини в таких мікропроцесорах не пов'язана з розрядністю шини даних. Характерним прикладом МП з роздільними шинами адрес і даних є мікропроцесорний комплект і8080, а з мультиплексованою шиною адреси і даних – мікропроцесорний комплект К588.

Контрольні питання

1. Опишіть типову структуру мікропроцесора

2. Дайте коротку характеристику основних регістрів мікропроцесора

3. Поясніть призначення акумулятора

4. У чому полягає призначення бітів регістра стану

5. Поясніть призначення та роботу стеку.

Лекція 3. Система команд та структура мікроконтролера

3.1. Формат команди та її виконання

З погляду системи команд і способів адресації операндів процесорне ядро сучасних 8-розрядних МК реалізує один із двох принципів побудови процесорів:

• процесори з CISC-архітектурою, які реалізують так звану повну систему команд (Complicated Instruction Set Computer);

• процесори з RISC-архітектурою, що реалізують скорочену систему команд (Reduced Instruction Set Computer).

CISC-процесори виконують великий набір команд із розвитими можливостями адресації, даючи виробнику можливість вибрати найбільше придатну команду для виконання необхідної операції. У застосуванні до 8-розрядних МК процесор з CISC-архітектурою може мати однобайтний, двохбайтний і трьохбайтний (рідко чотирьохбайтний) формат команд. При цьому система команд, як правило, неортогональна, тобто не всі команди можуть використовувати кожен зі способів адресації стосовно до кожного з регістрів процесора. Вибірка команди на виконання здійснюється побайтно протягом декількох циклів роботи МК. Час виконання команди може складати від 1 до 12 циклів. ДО МК із CISC-архітектурою відносяться МК фірми Intel з ядром MCS-51, які підтримуються в даний час цілим рядом виробників, МК сімейств HC05, HC08 і HC11 фірми Motorola і ряд інших.

У процесорах з RISC-архітектурою набір команд, які виконуються, скорочений до мінімуму. Для реалізації більш складних операцій приходиться комбінувати команди. При цьому всі команди мають формат фіксованої довжини (наприклад, 12, 14 або 16 біт), вибірка команди з пам'яті і її виконання здійснюється за один цикл (такт) синхронізації. Система команд RISC-процесора припускає можливість рівноправного використання всіх регістрів процесора. Це забезпечує додаткову гнучкість при виконанні ряду операцій. До МК із RISC-процесором відносяться МК AVR фірми Atmel, МК PIC16 і PIC17 фірми Microchip і інші.

Команди можуть складатися з 1, 2 або 3 байт, які послідовно розташовуються в пам'яті (рис.3.1).

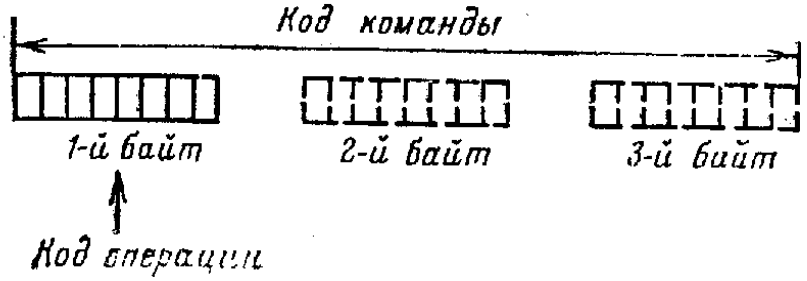


Рис.3.1. Формат команди.

Під час виконання попередньої команди лічильник команд містить значення, відповідне адресі того слова пам'яті, по якому розташовується перший байт наступної призначеної для виконання команди.

Перший байт команди відводиться для запису коду операції (КОП). Код операції указує, які дії мають бути виконані над даними, а також вид оброблюваних даних.

Таким чином, цикл виконання команди починається зі зчитування з пам'яті першого байта команди, який містить код операції.

Якщо відповідно до коду операції виявляється, що другий і третій байти команди разом утворюють адресу даних, призначених для обробки, то ці 2 байти мають бути переписані в ЦП. Після цього ЦП знає, де слід шукати необхідні дані.

Якщо ж код операції безпосередньо указує місце розташування даних, то їх обробка може починатися відразу після зчитування першого байта. Код операції може, наприклад, указувати, що обробці повинен піддатися другий байт коду команди. Виконання команди зводиться до виконання відповідних операцій над даними. При виконанні команди також може проводитися передача даних (наприклад, з пристрою введення в ЦП).

3.2 Команди асемблера

BRGE - Branch if Greater or Equal (Signed)

Description:

Conditional relative branch. Tests the Signed flag (S) and branches relatively to PC if S is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the signed binary number represented in Rd was greater than or equal to the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 4,k).

Operation:

(i) If Rd ≥ Rr (N ⊕ V = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRGE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k100

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            cp    r11,r12    ; Compare registers r11 and r12
            brge  greateq    ; Branch if r11 ≥ r12 (signed)
            ...
greateq:    nop              ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true

CP - Compare

Description:

This instruction performs a compare between two registers Rd and Rr. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - Rr

	Syntax:	Operands:	Program Counter:
(i)	CP Rd,Rr	$0 \le d \le 31, 0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0001	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet \overline{Rd7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
          cp    r4,r19    ; Compare r4 with r19
          brne  noteq     ; Branch if r4 <> r19
          ...
   noteq: nop             ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

Приклад коду на Ассемблері

```
.EQU      IN_PORT   =pind
.EQU      COMP      =2
.def      temp      =r16
.def      time_on   =r17

Loop1:
     dec  temp
     cpi  temp,0
     sbic IN_PORT,COMP
     brne Loop1
```

```
      mov   time_on,temp
      breq Loop2
      nop
Loop2:
```

Структурна схема мікроконтроллера ATMega328

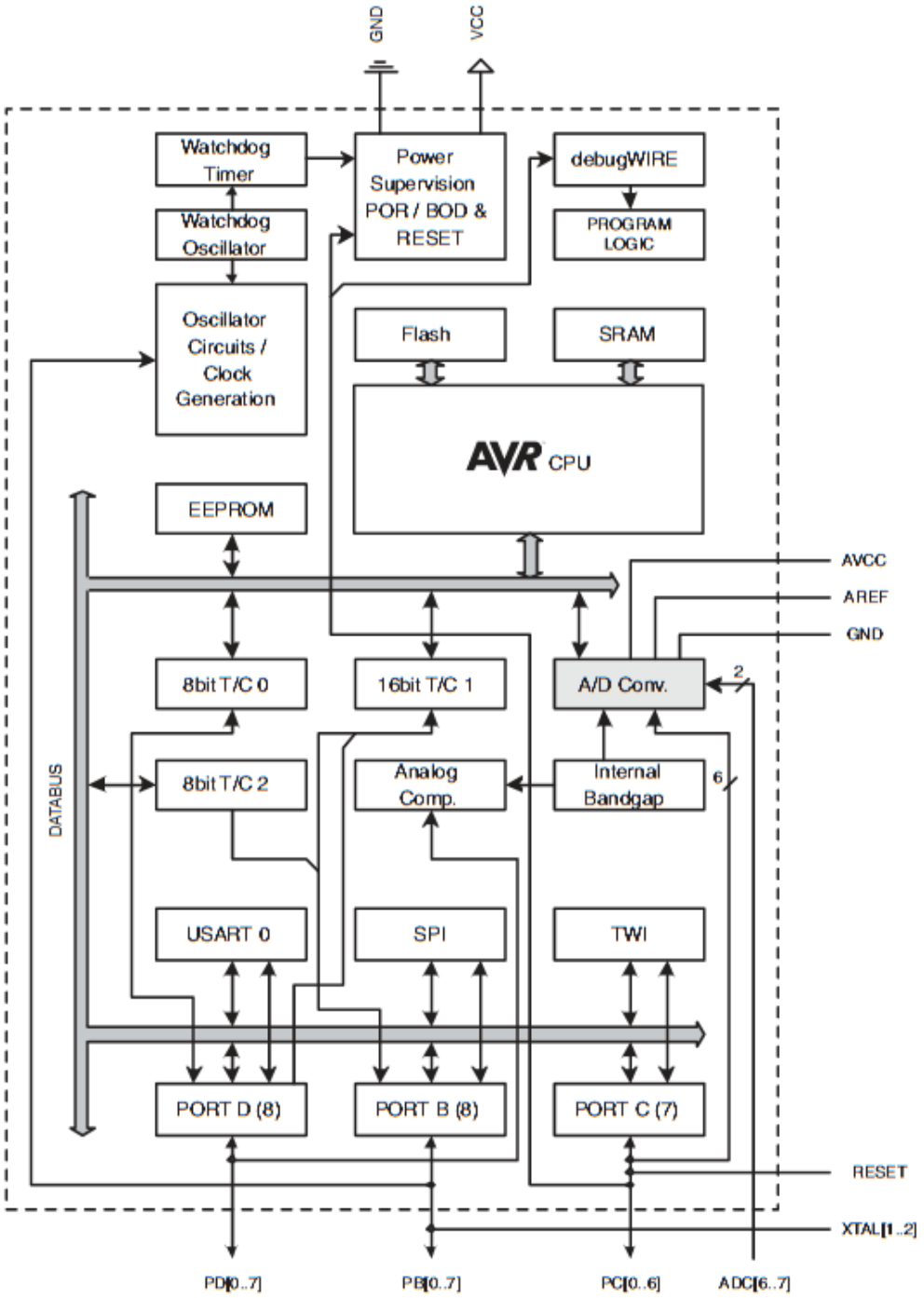


Рис. 4.5. - Структурна схема мікроконтроллера ATMega328

До складу мікроконтроллера ATMega328 входять *мікропроцесорне ядро*, *пам'ять* та *периферія*.

Пам'ять: постійна (Flash), оперативна (SRAM), постійна пам'ять даних (EEPROM),

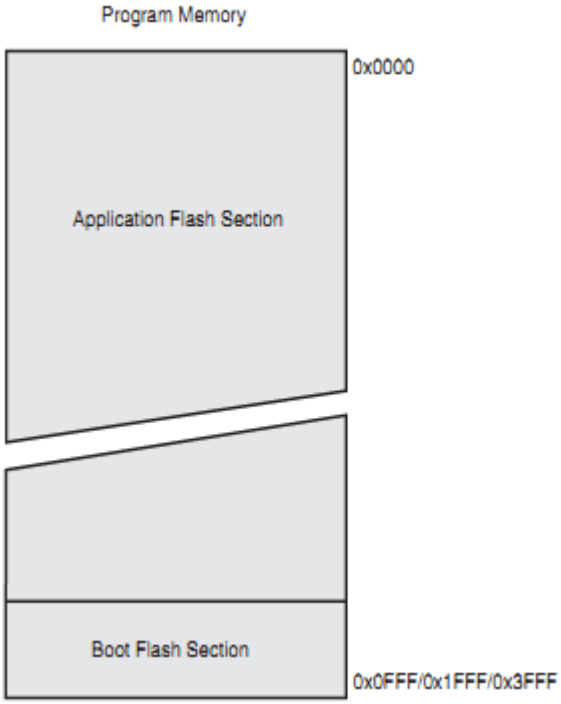


Рис. 4.6. – Пам’ять програм

Figure 8-3. Data Memory Map

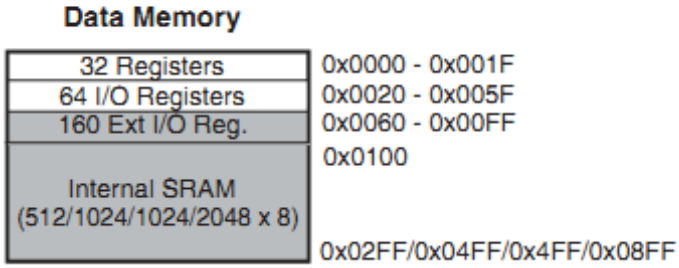


Рис. 4.7. – Пам'ять даних

Периферія:

- Схема формування сигналу ініціалізації (RESET)
- Блок контролю напруги живлення
- Сторожовий таймер
- Кола осцилятора та генератор тактових імпульсів
- Система переривань
- Порти вводу/виводу
- Лічильники-Таймери (сторожовий, ШІМ)
- Синхронний/асинхронний приймач-передавач
- 2-wire інтерфейс

- SPI інтерфейс
- АЦП
- Аналоговий компаратор

Схема формування сигналу ініціалізації (RESET) Відразу після виходу зі стану ініціалізації МК виконує наступні дії:

- запускає генератор синхронізації МК. Для стабілізації частоти тактування внутрішніми засобами формується затримка часу;
- завантажує в лічильник команд адресу початку робочої програми;
- проводить вибірку першої команди з пам'яті програм і приступає до виконання програми.

У ATMega328 лінія RESET є двонапрямленою і має низький активний рівень. При натисканні кнопки ініціалізації або увімкненні живлення буфер лінії встановлюється в режим вводу і реалізує так звану зовнішню ініціалізацію. МК може перейти в стан ініціалізації також за сигналами пристроїв контролю стану, які є в складі контролера. У цьому випадку говорять, що МК знаходиться в стані внутрішньої ініціалізації.

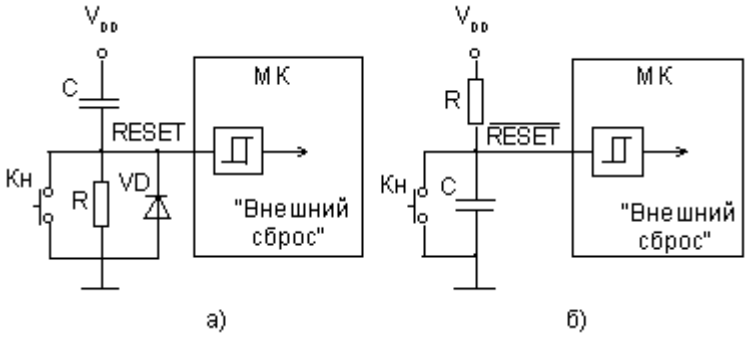


Рис. 3.10. Типові схеми формування сигналу зовнішньої ініціалізації

Блок контролю напруги живлення. У реальних умовах експлуатації може скластися така ситуація, при якій напруга живлення МК опуститься нижче мінімально допустимого, але не досягне порога відпускання схеми POR. У цих умовах МК може "зависнути". При відновленні напруги живлення до номінального значення МК залишиться непрацездатним. Для відновлення працездатності системи після "осідання" напруги живлення МК необхідно знову проініціалізувати (RESET). Для цієї мети реалізований додатковий блок детектування зниженої напруги живлення.

Сторожовий таймер. Якщо, незважаючи на усі вжиті заходи, МК усе-таки "завис", то на випадок виходу з цього стану є вмонтований модуль сторожового таймера. Принцип дії сторожового таймера показаний на Рис. 4.11.

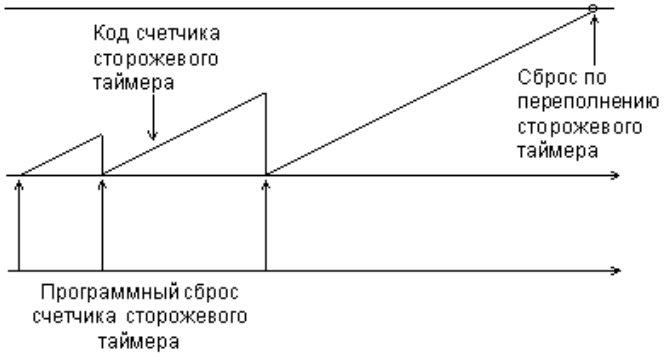


Рис. 3.11. Принцип дії сторожового таймера.

Основу сторожового таймера складає багаторозрядний лічильник. При скиданні МК лічильник встановлюється в нуль. Після переходу МК в активний режим роботи значення лічильника починає збільшуватися незалежно від виконуваної програми. При досягненні лічильником максимального коду генерується сигнал внутрішньої ініціалізації (RESET), і МК починає виконувати робочу програму спочатку.

Для вимкнення ініціалізації за переповненням сторожового таймера робоча програма МК повинна періодично скидати лічильник. Скидання лічильника сторожового таймера здійснюється шляхом виконання спеціальної команди #wdr.

Кола осцилятора та генератор тактових імпульсів. Сучасні МК містять вмонтовані тактові генератори, які вимагають мінімальної кількості зовнішніх часозадаючих елементів. На практиці використовуються три основних способи задавання тактової частоти генератора: за допомогою кварцового резонатора, керамічного резонатора і зовнішньої RC-ланки.

Типова схема підключення кварцового або керамічного резонатора наведена на Рис. 4.9а.

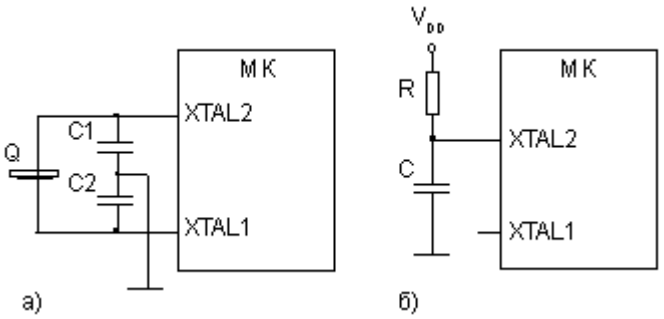


Рис. 3.12. Тактування МК з використанням кварцового або керамічного резонаторів (а) і з використанням RC-ланки (б).

Кварцовий або керамічний резонатор Q підключається до виводів XTAL1 і XTAL2, які зазвичай це вхід і вихід інвертуючого підсилювача. Номінали конденсаторів C1 і C2 визначаються виробником МК для конкретної частоти резонатора.

Використання кварцового резонатора дозволяє забезпечити високу точність і стабільність тактової частоти (розкид частот кварцового резонатора зазвичай складає менше 0,01%). Такий рівень точності потрібний для забезпечення точного ходу годинника реального часу або організації інтерфейсу з іншими пристроями. Основними недоліками кварцового резонатора є його низька механічна міцність (висока крихкість) і відносно висока вартість.

При менш жорстких вимогах до стабільності тактової частоти можливе використання більш стійких до ударного навантаження керамічних резонаторів. Багато керамічних резонаторів мають вмонтовані конденсатори, що дозволяє зменшити кількість зовнішніх елементів, що підключаються, від трьох до одного. Керамічні резонатори мають розкид частот порядку декількох десятих часток відсотка (звичайно близько 0,5%).

Найдешевшим способом задавання тактової частоти МК є використання зовнішньої RC-ланки, як показано на Рис. 4.9б, або внутрішньої RC-ланки. RC-ланка не забезпечує високої точності задавання тактової частоти (розкид частот може доходити до десятків відсотків). Це неприйнятно для багатьох задач, де потрібно точний підрахунок часу. Однак існує маса практичних задач, де точність задавання тактової частоти не має великого значення.

Контрольні питання

1. Принципи побудови процесорів з погляду системи команд і способів адресації операндів. Формат команди та її виконання

2. Опишіть структурну схему типового мікроконтролера на прикладі AVR контролера

3. Дайте коротку характеристику елементам периферії за призначенням

4. Призначення схеми формування сигналу ініціалізації AVR контролера

5. Призначення сторожового таймера AVR контролера

6. Організація роботи кіл осциляції мікроконтролера

Лекція 4. Програмно-апаратний комплекс Arduino

Arduino - це програмно-апаратний комплекс для вирішення прикладних завдань на основі AVR мікроконтролерів. Arduino представляє собою просту плату з мікроконтролером, а також спеціальне середовище розробки для написання програмного забезпечення.

Мова програмування Ардуіно запозичена з середовища програмування мультимедіа "Processing".

Ардуіно, як і інші аналогічні засоби (Parallax Basic Stamp, Raspberry, Netmedia's BX-24, Phidgets, MIT's Handyboard і ін.) Має на меті звільнити користувача від необхідності заглиблюватися в деталі внутрішнього устрою мікроконтролерів, надавши йому простий і зручний інтерфейс для їх програмування. Ардуіно на відміну від інших систем надає ряд переваг:

1. Просте і зручне середовище програмування. Середовище програмування Ардуіно зрозуміла і проста для початківців, але при цьому досить гнучка для просунутих користувачів. Вона заснована на середовищі програмування Processing, що може бути зручно для викладачів. Завдяки цьому, студенти, які вивчають програмування в середовищі Processing, зможуть легко освоїти Ардуіно.
2. Розширюване програмне забезпечення з відкритим вихідним кодом. Програмне забезпечення Ардуіно має відкритий вихідний код, завдяки цьому досвідчені програмісти можуть змінювати і доповнювати його. Можливості мови Ардуіно можна також розширювати за допомогою C ++ бібліотек. Завдяки тому, що він заснований на мові AVR C, просунуті користувачі, що бажають розібратися в технічних деталях, можуть легко перейти з мови Ардуіно на C або вставляти ділянки AVR-C коду безпосередньо в програми Ардуіно.
3. Розширюване відкрите апаратне забезпечення. Пристрої Arduino побудовані на базі мікроконтролерів Atmel ATmega8 і ATmega168. Завдяки тому, що всі схеми модулів Ардуіно опубліковані під ліцензією Creative Commons, досвідчені інженери і розробники можуть створювати свої версії пристроїв на основі існуючих. І навіть звичайні користувачі можуть збирати дослідні зразки Ардуіно для кращого розуміння принципів їх роботи і економії коштів.
4. Низька вартість. У порівнянні зі схожими апаратними платформами, плати Ардуіно мають відносно низьку вартість: готові модулі Ардуіно коштують не дорожче 50 $, а можливість зібрати плату вручну дозволяє максимально заощадити кошти і отримати Ардуіно за мінімальну ціну.
5. Кросплатформеність. Програмне забезпечення Ардуіно працює на операційних системах Windows, Macintosh OSX і Linux, в той час, як більшість подібних систем орієнтовані на роботу тільки в Windows.

Плати Ардуіно

Плати можна розділити на *контролери, шілди і аксесуари*. *Контролери* - це найважливіша частина - плата, яка містить мікроконтролер і в яку записується виконувана програма. *Шілд* - це плати розширення, які містять ту чи іншу периферію, керовану контролером. Шілд одягається зверху на контролер, утворюючи своєрідний "бутерброд".

Контролери Arduino Uno, Arduino Leonardo, Arduino Pro - пристрої на основі 8-розрядної мікроконтролера.

Arduino Due - це пристрій на основі мікропроцесора Atmel SAM3X8E ARM Cortex-M3. Це перша плата Ардуіно на базі 32-розрядного мікроконтролера ARM.

Завдяки використанню 32-розрядної ядра ARM, Arduino Due багато в чому перевершує типові плати на базі 8-розрядних мікроконтролерів. Найбільш суттєві відмінності полягають в наступному:

- 32-бітове ядро дозволяє обробляти 4х-байтові дані всього за один такт. Тактова частота - 84 МГц.
- Обсяг оперативної пам'яті SRAM складає 96 КБ.
- Обсяг флеш-пам'яті програм - 512 КБ.

Наявність DMA-контролера, що дозволяє розвантажити центральний процесор від виконання ресурсномістких операцій з пам'яттю.

Arduino YUN - це контролер з вбудованим WiFi модулем під управлінням ОС Linux і системою команд Ардуіно. Arduino YUN є комбінацією класичного Arduino Leonardo (на базі мікроконтролера ATmega32U4) і WiFi-системи на кристалі, що працює під управлінням Linino (дистрибутив ОС GNU / Linux на основі OpenWRT для мікропроцесорів MIPS).

Arduino Robot - перша офіційна версія Ардуіно, в конструкції якого передбачено колеса. Робот складається з двох плат, кожна з яких містить свій мікропроцесор. Плата приводів (Motor Board) контролює роботу двигунів, в той час, як керуюча плата (Control Board) зчитує показання датчиків і приймає рішення про подальші операції. Кожна з двох плат є повноцінне пристрій Ардуіно, програмований за допомогою середовища розробки Arduino IDE.

Arduino Esplora - це мікропроцесорний пристрій, спроектоване на основі Arduino Leonardo. Esplora відрізняється від усіх попередніх плат Arduino наявністю безлічі вбудованих, готових до використання датчиків для взаємодії. Esplora має вбудовані звукові і світлові індикатори (для виведення інформації), а також кілька датчиків (для введення інформації), таких, як джойстик, слайдер, датчик температури, акселерометр, мікрофон і світловий датчик.

Arduino ADK - це пристрій на основі мікроконтролера ATmega2560 (datasheet). У ньому реалізований USB-хост для підключення смартфонів на базі операційної системи Android.

Плати розширення: Arduino GSM, Arduino Ethernet, Arduino WiFi, Arduino Motor, Arduino Proto і т.д. + Різні аксесуари.

Arduino Nano

Arduino Nano - це повнофункціональний мініатюрний пристрій на базі мікроконтролера ATmega328 (Arduino Nano 3.0) або ATmega168 (Arduino Nano 2.x), адаптоване для використання з макетної платі. Arduino Nano розроблено і випускається фірмою Gravitech.

Характеристики:

- Мікроконтролер Atmel ATmega168 або ATmega328
- Робоча напруга (логічний рівень) 5В
- Напруга живлення (рекомендована) 7-12В
- Напруга живлення (гранична) 6-20В
- Цифрові входи / виходи 14 (з яких 6 можуть використовуватися як ШІМ-виходи)
- Аналогові входи 8
- Максимальний струм одного виведення 40 мА
- Flash-пам'ять 16 КБ (ATmega168) або 32 КБ (ATmega328) з яких 2 КБ використовуються загрузчиком
- SRAM 1 КБ (ATmega168) або 2 КБ (ATmega328)
- EEPROM 512 байт (ATmega168) або 1 КБ (ATmega328)
- Тактова частота 16 МГц
- Розміри плати 1.85 см x 4.3 см

Живлення

Arduino Nano може бути запитано через кабель Mini-B USB, від зовнішнього джерела живлення з нестабілізованою напругою 6-20В (через вивід 30) або зі стабілізованою напругою 5В (через вивід 27). Пристрій автоматично вибирає джерело живлення з найбільшим напругою.

Входи і виходи. З використанням функцій pinMode (), digitalWrite () і digitalRead () кожен з 14 цифрових висновків Arduino Nano може працювати в якості входу або виходу. Робоча напруга виводів - 5В. Максимальний струм, який може віддавати або споживати один вивід, становить 40 мА. Всі виводи пов'язані з внутрішніми підтягуються резисторами (за замовчуванням відключеними) номіналом 20-50 кОм. Крім основних, деякі виводи Ардуіно можуть виконувати додаткові функції:

Послідовний інтерфейс: виводи 0 (RX) і 1 (TX). Використовуються для отримання (RX) і передачі (TX) даних по послідовному інтерфейсу. Ці виводи з'єднані з відповідними висновками мікросхеми-перетворювача USB-UART від FTDI.

Зовнішні переривання: виводи 2 і 3. Дані виводи можуть бути налаштовані в якості джерел переривань, що виникають при різних умовах: при низькому рівні сигналу, по фронту, по спаду або при зміні сигналу. Для отримання додаткової інформації див. Функцію attachInterrupt ().

ШІМ: виводи 3, 5, 6, 9, 10 і 11. За допомогою функції analogWrite () можуть виводити 8-бітові аналогові значення в вигляді ШІМ-сигналу.

Інтерфейс SPI: виводи 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Дані виводи дозволяють здійснювати зв'язок по інтерфейсу SPI. У пристрої реалізована апаратна підтримка SPI, проте на даний момент мова Ардуіно поки її не підтримує.

Світлодіод: вивід 13. Вбудований світлодіод, приєднаний до цифрового виводу 13. При відправці значення HIGH світлодіод включається, при відправці LOW - вимикається.

I2C: виводи 4 (SDA) і 5 (SCL). З використанням бібліотеки Wire (документація на веб-сайті Wiring) дані виводи можуть здійснювати зв'язок по інтерфейсу I2C (TWI).

Крім перерахованих на платі існує ще кілька виводів:

- *AREF*. Опорна напруга для аналогових входів. Може бути задіяний функцією analogReference ().
- *Reset*. Формування низького рівня (LOW) на цьому висновку призведе до перезавантаження мікроконтролера. Зазвичай цей вивід служить для функціонування кнопки скидання на платах розширення.

Середовище розробки Arduino

Інсталяція:

- Інсталяція програми
- Інсталяція драйвера (тільки з підключеною платою через USB)

Середовище розробки Arduino складається з вбудованого текстового редактора програмного коду, області повідомлень, вікна виведення тексту (консолі), панелі інструментів з кнопками часто використовуваних команд і

декількох меню. Для завантаження програм і зв'язку середовище розробки підключається до апаратної частини Arduino.

Програма, написана в середовищі Arduino, називається *скетч*. Скетч пишеться в текстовому редакторі, що має інструменти вирізки / вставки, пошуку / заміни тексту. Під час збереження і експорту проекту в області повідомлень з'являються пояснення, також можуть відображатися виникли помилки. Вікно виведення тексту (консоль) показує повідомлення Arduino, що включають повні звіти про помилки та іншу інформацію. Кнопки панелі інструментів дозволяють перевірити і записати програму, створити, відкрити і зберегти скетч, відкрити моніторинг послідовної шини.

Блокнот (Sketchbook)

Середовищем Arduino використовується принцип блокнота: стандартне місце для зберігання програм (скетчів). Скетчі з блокнота відкриваються через меню File> Sketchbook або кнопкою Open на панелі інструментів. При першому запуску програми Arduino автоматично створюється директорія для блокнота. Розташування блокнота змінюється через діалогове вікно Preferences.

Закладки, Файли і Компіляція

Дозволяють працювати з декількома файлами скетчів (кожен відкривається в окремій закладці). Файли коду можуть бути стандартними Arduino (без розширення), файлами C (розширення * .c.), файлами C ++ (* .cpp) або файлами заголовків (.h).

Завантаження скетчу в Arduino

Перед завантаженням скетчу потрібно задати необхідні параметри в меню Tools> Board і Tools> Serial Port. Платформи описуються далі по тексту. В ОС Windows порти можуть позначатися як COM1 або COM2 (для плати послідовної шини) або COM4, COM5, COM7 і вище (для плати USB). Визначення порту USB проводиться в поле Послідовною шини USB Диспетчера пристроїв Windows.

Після вибору порту і платформи необхідно натиснути кнопку завантаження на панелі інструментів або вибрати пункт меню File> Upload to I / O Board. Сучасні платформи Arduino перезавантажуються автоматично перед завантаженням. На старих платформах необхідно натиснути кнопку перезавантаження. На більшості плат під час процесу будуть мигати світлодіоди RX і TX. Середовище розробки Arduino виведе повідомлення про закінчення завантаження або про помилки.

При завантаженні скетчу використовується Завантажувач (Bootloader) Arduino, невелика програма, що завантажується в мікроконтролер на платі. Вона дозволяє завантажувати програмний код без використання додаткових апаратних засобів. Завантажувач (Bootloader) активний протягом декількох секунд при перезавантаженні платформи і при завантаженні будь-якого з

скетчів в мікроконтролер. Робота завантажувач (Bootloader) розпізнається по миготінню світлодіода (13 пін) (напр.: при перезавантаженні плати).

Библиотекі

Бібліотеки додають додаткову функціональність скетчам, наприклад, при роботі з апаратною частиною або при обробці даних. Для використання бібліотеки необхідно вибрати меню Sketch> Import Library. Одна або кілька директив #include будуть розміщені на початку коду скетчу з подальшою компіляцією бібліотек і разом зі скетчем. Завантаження бібліотек вимагає додаткового місця в пам'яті Arduino. Невикористані бібліотеки можна видалити з скетчу прибравши директиву #include.

На Arduino.cc є список бібліотек. Деякі бібліотеки включені в середу розробки Arduino. Інші можуть бути завантажені з різних ресурсів. Для установки викачаних бібліотек необхідно створити директорію "libraries" в папці блокнота і потім розпакувати архів. Наприклад, для установки бібліотеки DateTime її файли повинні знаходиться в папці / libraries / DateTime папки блокнота.

Мова Ардуіно

Мова Arduino має чотири складових:

- оператори
- дані
- функції
- бібліотекі

оператори

- setup ()
- loop ()
- оператори мови С

дані: типи даних з мови С

функції:

- цифрове введення / виведення
- аналогове введення / виведення
- час
- математичні обчислення
- тригонометрія
- випадкові числа
- біти і байти
- зовнішні переривання
- переривання

бібліотеки:

- EEPROM
- SD
- SPI
- SoftwareSerial
- Wire
- допоміжні класи

    - клас Serial
    - клас Stream

Системі числення:

- двійкова
- десяткова
- шістнадцяткова.

Шістнадцяткова – 0..15 – 8 біт = 1 байт

Для того, щоб перетворити від'ємне десяткове число в двійкову форму, необхідне це число записати як позитивне в двійковому вигляді. Після цього записати обернений код двійкового числа, а потім додати до оберненого коду двійкового числа одиницю, внаслідок чого отримаємо додатковий код двійкового числа, який дорівнюватиме від'ємному десятковому числу. Розглянемо наступний приклад. Представити в двійковому вигляді від'ємне число – 77. Для цього отримаємо двійкове число додатного числа 77.

77 = 01001101

Обернений код цього числа отримаємо заміною 0 на 1, і 1 на 0.

10110010

Для отримання додаткового коду додамо до оберненого коду 1.

```
 10110010 – обернений код
+        1
 10110011 – додатковий код числа 77
```

Коротка довідка про типи даних Arduino

void - Ключове слово void використовується тільки при оголошенні функцій. Воно вказує на те, що оголошується функція не повертає ніякого значення тієї функції, з якої вона була викликана.

Boolean-Змінні типу boolean можуть приймати одне з двох значень: true або false. (Кожна змінна типу boolean займає в пам'яті один байт.)

char -Тип даних, який займає в пам'яті 1 байт і зберігає символьне значення. Символи пишуться в одинарних лапках, наприклад: 'A' (сукупність символів - рядки - пишуться в двійнят лапках: "ABC").

unsigned char -Те ж саме, що і тип даних *byte*. Беззнаковий тип даних, що займає в пам'яті 1 байт.

int - цілочисельний тип даних. Це основний тип даних для зберігання чисел.

В Arduino Uno (і інших платах на базі 8-ти бітних мікроконтролерів) змінні типу int зберігають 16-бітові (2-байтові) значення. Така розмірність дає діапазон від -32768 до 32767 (мінімальне значення -2 ^ 15 і максимальне значення (2 ^ 15) -1).

unsigned int - в 8-ти бітних мікроконтролерів, змінні типу unsigned int (беззнакові цілі) містять двобайтові значення. Відмінність полягає в тому, що замість негативних чисел вони можуть зберігати лише позитивні значення в зручному діапазоні від 0 до 65535 ((2 ^ 16) -1).

Long - Змінні типу long володіють розширеним розміром для зберігання чисел і мають розмірність 32 біта (4 байта), що дозволяє їм зберігати числа в діапазоні від -2 147 483 648 до 2 147 483 647.

unsigned long - мають розмірність 32 біта (4 байта). Змінні типу unsigned long, на відміну від звичайного long, зберігають тільки позитивні числа в діапазоні від 0 до 4 294 967 295 (2 ^ 32 - 1).

short - це 16-бітний тип даних. На всіх платах Ардуіно

float - Тип даних для чисел з плаваючою точкою (чисел з десятковим роздільником). Числа з плаваючою точкою часто використовуються для подання аналогових або безперервних величин, оскільки дають можливість окреслити їх більш точно, ніж цілі числа. Числа з плаваючою точкою є 32 біта (4 байта) інформації і можуть досягати величезних значень від -3.4028235E + 38 до 3.4028235E + 38.

Точність дрібних чисел типу float становить 6-7 десяткових знаків. Мається на увазі загальна кількість цифр, а не кількість знаків після коми.

double - для 8-ти бітних мікроконтролерів змінні типу double займають 4 байта. Аналогічні змінним float.

В Arduino Duo (32-х бітний) змінні double мають точність 8 байт (64 біта).

Створення (оголошення) *массиву*. Масив - це область пам'яті, де можуть послідовно зберігатися кілька значень.

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
```

```
char message[6] = "hello";
```

string - Текстовий рядок. Може бути оголошена двома способами: можна використовувати тип даних String, який входить в ядро, починаючи з версії 0019; або оголосити рядок як *масив символів* char з нульовим символом в кінці.

```
char Str1 [15];
char Str2 [8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3 [8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o', '\ 0'};
char Str4 [] = "arduino";
char Str5 [8] = "arduino";
char Str6 [15] = "arduino";
```

Рядки завжди оголошуються в подвійних лапках ("Abc"), а символи завжди оголошуються в одинарних лапках ('A').

Контрольні питання

1. Дайте короткий опис програмно-апаратний комплексу Arduino. Яки його переваги перед аналогами.

2. Основні компоненти середовища розробки Arduino. Дайте їх короткий опис.

3. Дайте характеристику складовим мови Arduino.

4. Короткий опис типів даних Arduino

Лекція 5. Мова С++. Програмування в Arduino.

Оголошення змінної

Оголошення змінної в С ++ відбувається таким чином: спочатку вказується тип даних для цієї змінної а потім назва цієї змінної. В С ++ оператор присвоювання (=) - не є знаком рівності і не може використовуватися для порівняння значень. Оператор рівності записується як «подвійне одно» - ==. Присвоєння використовується для збереження певного значення в змінній. Наприклад, запис виду a = 10 задає змінної а значення числа 10.

Цикли

Якщо ми знаємо точну кількість дій (ітерацій) циклу, то можемо використовувати цикл *for*. Синтаксис його виглядає приблизно так:

for (дію до початку циклу; умова продовження циклу; дії в кінці кожної ітерації циклу)
{
	інструкція циклу;
	інструкція циклу 2;
	інструкція циклу N;
}
Ітерацією циклу називається один прохід цього циклу

Коли ми не знаємо, скільки ітерацій повинен зробити цикл, нам знадобиться цикл *while* або *do ... while*. Синтаксис циклу while в С ++ виглядає наступним чином.

while (Умова)
 {
    Тіло циклу;
}
Даний цикл буде виконуватися, поки умова, вказане в круглих дужках є істиною.

Конструкція розгалуження в С ++

Оператор if служить для того, щоб виконати будь-яку операцію в тому випадку, коли умова є вірною. Умовна конструкція в С ++ завжди записується в круглих дужках після оператора if.

Усередині фігурних дужок вказується тіло умови. Якщо умова виконається, то почнеться виконання всіх команд, які знаходяться між фігурними дужками.

Оператор else. Кожному оператору if відповідає тільки один оператор *else.* Сукупність цих операторів - else if означає, що якщо не виповнилося попереднє умова, то перевірити дане. Якщо жодна з умов не вірно, то виконується тіло оператора else.

Оператори порівняння
== (дорівнює)
! = (Не дорівнює)
<(Менше ніж)
> (Більше ніж)
<= (Менше або дорівнює)
> = (Більше або дорівнює)

Логічні оператори
&& (І)
|| (АБО)
! (НЕ)

Бітові оператори
& (Побітовое І)
| (Побітовое АБО)
^ (Побітовое XOR або виключає АБО)
~ (Побітовое НЕ)
<< (побітовий зрушення вліво)
>> (побітовий зрушення вправо)

Складні оператори
++ (інкремент)
-- (декремент)
+ = (Складене додавання)
- = (складене віднімання)
* = (Складене множення)
/ = (Складене ділення)
& = (Складене побітовое І)
| = (Складене побітовое АБО)

Функції
Функції - один з найважливіших компонентів мови С ++.
Будь-яка функція має тип, також, як і будь-яка змінна.
Функція може повертати значення, тип якого аналогічний типу самої функції.
Якщо функція не повертає ніякого значення, то вона повинна мати тип **void**
(такі функції іноді називають процедурами)

При оголошенні функції, після її типу має перебувати ім'я функції і дві круглі дужки - відкриває і закриває, всередині яких можуть знаходитися один або кілька аргументів функції, яких також може не бути взагалі.
Після списку аргументів функції ставиться фігурна дужка, що відкриває, після якої знаходиться саме тіло функції.
В кінці тіла функції обов'язково ставиться фігурна дужка, що закриває.

Arduino

Функції setup (), loop ()

Під час виклику функції *setup* (), ініціалізує і встановлює початкові значення. Функція *setup* () викликається, коли стартує скетч. Використовується для ініціалізації змінних, визначення режимів роботи виводів, запуску використовуваних бібліотек і т.д. Функція setup запускає тільки один раз, після кожної подачі живлення або скидання плати Arduino.

Функція *loop* () забезпечує нескінченний робочий цикл програми. У циклі виконується опитування стану виводів, зміна їх стану, прийом-передача даних робота з АЦП та ін.

приклад:

```
int buttonPin = 3;
void setup()
{
  // put your setup code here, to run once:
}
void loop()
{
  // ...
}
```

Контрольні питання

1. Наведіть приклади оголошення змінної, та організації циклу

2. Наведіть приклади використання операторів разголуження, порівняння, бітових та логічних операторів

3. Наведіть приклади використання функцій

4. Особливості використання функцій *setup () і loop ()*

Лекція 6. Цифрове введення / виведення

Мікроконтроллер ATMega328 має деяку кількість ліній вводу/виводу, що об'єднані в 8-розрядні паралельні порти вводу/виводу. У пам'яті МК кожному порту вводу/виводу відповідає своя адреса регістра даних (див. малюнок 2.6). Звертання до регістра даних порту вводу/виводу відбувається тими ж командами, що і звертання до пам'яті даних. Крім того, окремі розряди портів можуть бути опитані або встановлені побітово.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–	
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–	
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	Reserved	–	–	–	–	–	–	–	–	
0x0D (0x2D)	Reserved	–	–	–	–	–	–	–	–	
0x0C (0x2C)	Reserved	–	–	–	–	–	–	–	–	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	92
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	92
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	92
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	91
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	91
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	92
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	91
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	91
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	91
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–	

Рис. 2.6. – Адресація регістрів вводу-виводу

Основні особливості:

- двонаправлені порти, напрямок передачі яких (ввід або вивід) визначається в процесі ініціалізації МК;
- порти з альтернативною функцією. Окремі лінії цих портів використовуються спільно з вбудованими периферійними пристроями, такими як таймери, АЦП, контролери послідовних інтерфейсів та ін.

Регістри керування портами вводу-виводу `PORTx, DDRx`

Порт може бути сконфігуровано (З картинки)

- на вхід;
- на вхід з підтяжкою;
- на вихід.

Ініціалізація порту вводу-виводу

На Ардуіно:

pinMode (pin, mode)
Параметри
pin: номер виводу, режим роботи якого задається.

mode: приймає значення INPUT, OUTPUT або INPUT_PULLUP
Значення, що повертаються - немає

digitalWrite (pin, value)
Параметри
pin: номер виводу
value: значення HIGH або LOW
Значення, що повертаються - немає

digitalRead (pin)
Параметри pin: номер цифрового виводу, з якого необхідно вважати значення (int)
Значення, що повертаються HIGH або LOW
…

На С

```
unsigned char i;
/* Define pull-ups and set outputs high */
/* Define directions for port pins */
PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
/* Insert nop for synchronization*/
__no_operation();
/* Read port pins */
i = PINB;
...
```

На ассемблері

```
; Define pull-ups and set outputs high
; Define directions for port pins
Ldi   r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi   r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out   PORTB,r16
out   DDRB,r17
; Insert nop for synchronization
nop
; Read port pins
in r16,PINB
...
```

Контрольні питання

1. Які особливості портів вводу/виводу для мікроконтроллерів AVR
2. Як можна сконфігурувати порт вводу/виводу
3. Як ініціалізувати порт вводу/виводу у середовищі Arduino
4. Порівняти процедури ініціалізації в Arduino, на С та на асемблері

Лекція 7. Функції часу

Функції часу використовують для формування і виміру часових інтервалів.

delay (ms)

Параметри: ms - кількість мілісекунд, на які необхідно призупинити програму (unsigned long)

Значення, що повертаються - немає

Опис: Припиняє виконання програми на вказаний проміжок часу (в мілісекундах). (В 1 секунді - 1000 мілісекунд.)

delayMicroseconds (us)

Параметри: us - кількість мікросекунд, на які необхідно призупинити програму (unsigned int)

Значення, що повертаються - немає

Опис: Припиняє виконання програми на вказаний проміжок часу (в мікросекундах).

На даний момент найбільша кількість, що дозволяє сформувати точну затримку, - 16383. В майбутніх версіях Ардіуно цей показник може бути змінений. Для створення затримок тривалістю більше, ніж кілька тисяч мікросекунд, використовуйте функцію *delay ()*.

millis ()

Параметри - Hi

Значення, що повертаються - Кількість мілісекунд, що пройшли з моменту старту програми (unsigned long)

Опис: Повертає кількість мілісекунд, що пройшли з моменту старту програми Ардуіно. Повертається число переповниться (скинеться в 0) через приблизно 50 днів.

micros()

Параметри - Hi

Значення, що повертаються - Кількість мікросекунд, що минули з моменту старту програми (unsigned long)

Опис: Повертає кількість мікросекунд, що минули з моменту початку виконання програми Arduino. Повертається число переповниться (скинеться в 0) через приблизно 70 хвилин. На платах Arduino з тактовою частотою 16 МГц (Duemilanove і Nano) роздільна здатність цієї функції становить чотири мікросекунди (тобто повертається значення буде завжди кратно чотирьом). На платах Ардуіно з тактовою частотою 8 МГц (LilyPad), роздільна здатність функції становить вісім мікросекунд.

pulseIn (pin, value)

pulseIn (pin, value, timeout)

Параметри: pin - номер виводу, з якого необхідно вважаючи імпульс (int)

value: тип зчитує імпульсу: HIGH або LOW (int)

timeout (опціонально): час очікування імпульсу в мікросекундах; значення за замовчуванням - одна секунда (unsigned long)

Значення, що повертаються - тривалість імпульсу (в мікросекундах) або 0 в разі відсутності імпульсу протягом таймаута (unsigned long)

Опис: Зчитує тривалість імпульсу (будь-якого - HIGH або LOW) на виведення. Наприклад, якщо задане значення (value) - HIGH, то функція PulseIn () очікує появи на виведення сигналу HIGH, потім засікає час і чекає перемикати в стан LOW, після чого зупиняє відлік часу. Функція повертає тривалість імпульсу в мікросекундах, або 0 в разі відсутності імпульсу протягом певного часу очікування.

Емпіричним шляхом встановлено, що при використанні функції для вимірювання широких імпульсів можливе виникнення помилок. Функція працює з імпульсами тривалістю від 10 мікросекунд до 3 хвилин.

```
unsigned long time;
void setup () {
  Serial.begin (9600);
}
void loop () {
  Serial.print ( "Time:");
  time = millis ();
  // Виводимо час з моменту старту програми
  Serial.println (time);
  // Чекаємо 1 секунду, щоб не відправляти великий масив даних
  delay (1000);
}
```

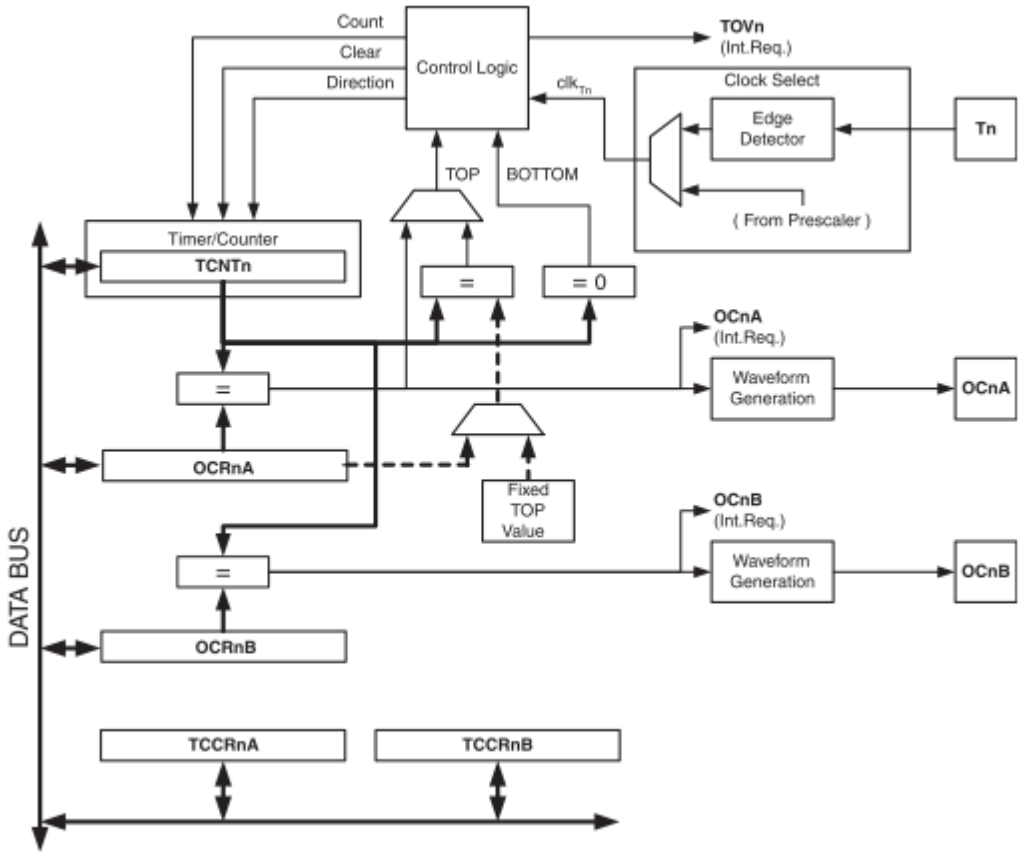
Таймери/лічильники як основні елементи для реалізації функцій часу

У ATMega328 передбачені три таймери/лічильника загального призначення: два 8-розрядних і один 16-розрядний. Кожний з таймерів індивідуально підключається до одного з виходів 10-розрядного попереднього подільника частоти. Всі таймери можуть використовуватися як таймери з внутрішнім джерелом імпульсів чи як лічильники імпульсів, що надходять ззовні. Як джерело імпульсів для таймерів можна вибрати сигнал тактовою частотою мікроконтролера (СК), імпульси попереднього подільника (СК/8, СК/64, СК/256 чи СК/1024) чи імпульси з відповідного зовнішнього виводу. Крім того, таймери можуть бути зупинені.

Основні регістри таймера/лічильника:

- лічильний регістр TCNTx;
- регістри порівняння з виходом (Output Compare Registers) OCRxA and OCRxB;
- регістр маски переривання таймера (Timer Interrupt Mask Register) TIMSK0;
- регістр прапорців переривання таймера (Timer Interrupt Flag Register) TIFR0.

Figure 15-1. 8-bit Timer/Counter Block Diagram



Контрольні питання

1. Призначення та опис функції часу у середовищі Arduino

2. Які обмеження існують при використанні функції часу у середовищі Arduino

3. Призначення таймерів/лічильників мікроконтроллера ATMega328

4. Призначення регістрів таймерів/лічильників мікроконтроллера ATMega328

Лекція 8. Асинхронний послідовний обмін

Найбільш розповсюджена форма послідовного зв'язку — асинхронний обмін, при якому байт даних посилається як пакет, що включає інформацію про початок і кінець передавання даних, а також інформацію для контролю помилок.

Першим передається не біт даних, а старт-біт, що вказує на початок передавання даних (початок пакета). Цей біт використовується приймачем для синхронізації процесу читання даних, що передаються за старт-бітом (молодший біт даних йде першим). Після бітів даних може випливати біт парності (контрольний біт), що використовується для перевірки правильності отриманих даних. Існує два типи перевірки на парність. Перевірка на непарність («odd») означає, що число одиниць у пакеті даних, включаючи біт парності, повинно бути непарним (наприклад, 0x55 буде мати біт парності рівним 1, щоб зробити число одиничних бітів рівним п'яти, тобто непарним). Перевірка на парність («even»), навпаки, означає що число одиничних бітів повинно бути парним (наприклад, при передаванні числа 0x55 біт парності дорівнює 0).

За бітом парності передається стоп-біт, що використовується приймачем для обробки кінця передавання пакета.

Асинхронний пакет даних показаний на рис. 2.28. Існує набір параметрів, що повинний бути відомий при реалізації обміну. Одним з таких параметрів є число переданих бітів даних, що визначається типом приймального і передавального пристроїв. Пакет на рис. 2.28 містить тільки 5 біт даних (таке число бітів використовувалося в телетайпах), але можливі пакети довжиною до 8 біт.

Поряд з бітами парності («odd») чи непарності («even») можливі інші варіанти контрольних бітів: «no», «mark» і «space». «no» означає відсутність біта парності в пакеті. «mark» чи «space» означає, що замість біта парності завжди посилається „1" («mark») чи „0" («space»), відповідно. Ці варіанти контрольних бітів використовуються досить рідко - у тих випадках, коли необхідно дати приймачу додатковий час на оброблення пакета.

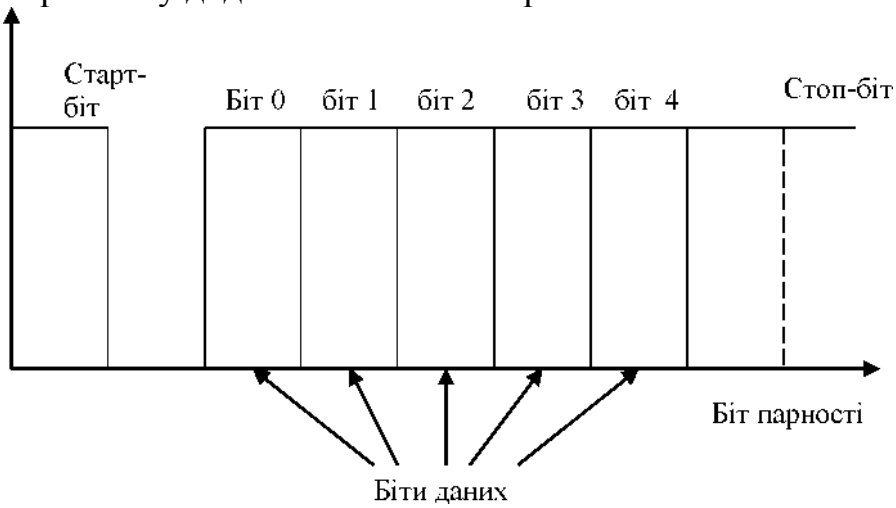


Рисунок 2.28 - Асинхронне послідовне передавання даних

Кількість стоп-бітів також може бути різна. Другий стоп-біт може вводитися для тієї ж мети, що і контрольні біти «mark» і «space» - щоб дати приймачу більше часу для обробки прийнятого пакета.

Практично всі сучасні пристрої використовують для асинхронного обміну формат даних «8-N-1», що означає передачу 8 біт даних, відсутність біта парності й один стоп-біт. Біт парності і додатковий стоп-біт, звичайно, не потребуються для послідовного зв’язку.

Найбільш популярний протокол асинхронного послідовного зв’язку називається „RS-232”, що у наш час є міжнародним стандартом. Це дуже старий стандарт, використовуваний для зв’язку комп’ютерів.

Організація обміну даними між платою Arduino і комп'ютером через USB

Для зв'язку з платою Arduino можна використовувати спеціальну програму моніторингу послідовного порту (Serial Monitor), вбудовану в програмне забезпечення Ардуіно. Монітор послідовної шини відображає дані посилаються в платформу Arduino (плата USB або плата послідовної шини). Для відправки даних вибирається швидкість передачі зі списку, відповідна значенню Serial.begin в скетчі. Потім необхідно ввести текст і натиснути кнопку Send або Enter.

Плата Arduino Nano має один послідовний порт (також відомий як UART або USART): Serial. Він пов'язаний з цифровими виводами 0 (RX) і 1 (TX), а також використовується для зв'язку з комп'ютером через USB. Таким чином, під час використання послідовного порту, виводи 0 і 1 не можуть використовуватися в якості цифрових входів або виходів.

Для забезпечення зв'язку плати Ардуіно з комп'ютером або іншими пристроями використовується клас Serial. Клас - це абстрактний тип даних. За допомогою класу описується деяка сутність (її характеристики і можливі дії). Наприклад, клас може описувати змінні, параметри і функції послідовного порту. Клас Serial містить близько 20 функцій. Розглянемо деякі.

Функції для роботи з послідовним портом плати Ардуіно

if (Serial)

Параметри - Ні. **Значення, що повертаються** - boolean: повертає true, якщо вказаний послідовний порт готовий до роботи.

Опис: Дозволяє перевірити готовність певного послідовного порту.

Serial.begin(speed)
Serial.begin(speed, config)
Параметри:
speed: швидкість в бітах на секунду (бодах) - long
*config***:** задає кількість біт даних, перевірку парності и стопові біти.
SERIAL_5N1

SERIAL_6N1
SERIAL_7N1
SERIAL_8N1 (за замовченням)
SERIAL_5N2
SERIAL_8E2
SERIAL_7O2

Опис: Задає швидкість передачі даних по послідовному інтерфейсу в бітах в секунду (бодах). Для взаємодії з комп'ютером слід використовувати одну з попередньо встановлених швидкостей обміну: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 або 115200.

Другий аргумент цієї функції необов'язковий. Він дозволяє налаштувати кількість біт даних, перевірку парності і степових біти. За замовчуванням, посилка складається з 8 біт даних, без перевірки парності, з одним стоповим бітом.

Serial.end ()

Параметри - Ні. **Значення, що повертаються** - немає

Опис: Функція розриває послідовну зв'язок, після чого висновки RX і TX знову можуть використовувати як висновки загального призначення. Для відновлення послідовного з'єднання необхідно використовувати функцію *Serial.begin ().*

Serial.available()

Параметри – Нема. **Значення, що повертаються**: кількість байт, доступних для зчитування

Опис: Повертає кількість байт (символів) доступних для зчитування з буфера послідовного порту. Під символами розуміються дані, які вже прийняті і зберігаються в послідовному приймальному буфері (який може зберігати максимум 64 байта).

Serial.read()

Параметри – Нема. **Значення, що повертаються**: Перший байт прийнятих даних (або -1, если таких нема) - int

Опис: Зчитує дані, що надходять по послідовному інтерфейсу.

Serial.print(val)

Serial.print(val, format)

Параметри:

*val***:** значення, яке необхідно вивести - будь-який тип даних

*format***:** визначає систему числення (для цілочисельних типів), а також кількість десяткових знаків після коми (для чисел з плаваючою точкою).

Значення, що повертаються:

size_t (long): функція print () повертає кількість виведених байт. Зчитування цього значення не обов'язково.

Опис: Функція виводить через послідовний порт заданий ASCII текст у вигляді, зрозумілому для людини. Ця команда може мати кілька різних форм.

При виведенні числа кожній його цифрі відповідає один ASCII-символ. Дробові числа теж виводяться у вигляді ASCII-цифр, при цьому після коми за замовчуванням залишається два десяткових знака. Байти виводяться у вигляді окремих символів, а символи і рядки виводяться без змін - "як є". Наприклад:

Serial.print(78) - виведе "78"
Serial.print(1.23456) - виведе "1.23"
Serial.print('N') - виведе "N"
Serial.print("Hello world.") - виведе "Hello world."

Другий параметр необов'язковий. Він задає формат виведення; цей параметр може набувати таких значень: BIN (двійкова система з основою 2), OCT (восьмерична система з основою 8), DEC (десяткова система з основою 10), HEX (шістнадцятирічна система з основою 16). Для числі з плаваючою точкою цей параметр визначає кількість десяткових знаків після коми. наприклад:

Serial.print (78, BIN) - виведе "1001110"
Serial.print (78, OCT) - виведе "116"
Serial.print (78, DEC) - виведе "78"
Serial.print (78, HEX) - виведе "4E"
Serial.println (1.23456, 0) - виведе "1"
Serial.println (1.23456, 2) - виведе "1.23"
Serial.println (1.23456, 4) - виведе "1.2346"

Serial.println (val)
Serial.println (val, format)
параметри:
val: значення, яке необхідно вивести - будь-який тип даних
format: визначає систему числення (для цілочисельних типів), а також кількість десяткових знаків після коми (для чисел з плаваючою точкою).
Значення, що повертаються:
size_t (long): функція println () повертає кількість виведених байт. Зчитування цього значення не обов'язково.
Опис: Виводить через послідовний порт ASCII-текст в зрозумілому для людини вигляді з символами повернення каретки (ASCII 13 або '\ r') і нового рядка (ASCII 10 або 'n'). Ця команда має такі ж форми, як і Serial.print ().

Приклад коду для роботи з послідовним портом.

```
int incomingByte = 0;
void setup()
{
    Serial.begin(9600);
}
void loop()
{
    if (Serial.available() > 0) {
```

```
            incomingByte = Serial.read();
            Serial.print("I received: ");
            Serial.println(incomingByte, DEC);
        }
    }
```

Контрольні питання

1. Як організовано асинхронний послідовний обмін? Структура пакету даних

2. Як організовано обмін даними між платою Arduino і комп'ютером через USB

3. Опишіть функції ініціалізації послідовного порту і перевірки прийнятих даних.

4. Опишіть основні функції прийому - передачі даних через послідовний порт

Лекція 9. Переривання

Переривання - це сигнали, що переривають нормальний перебіг програми. Вони зазвичай використовуються для апаратних пристроїв, що вимагають негайної реакції на появу подій.

Обробка переривань у мікроконтроллера (МК) відбувається за допомогою модуля переривань, який приймає запити переривання й організовує перехід до виконання визначеної програми. Запити переривання можуть надходити як від зовнішніх джерел, так і від джерел, розташованих у різних внутрішніх модулях мікроконтроллера.

Як входи для прийому запитів від зовнішніх джерел найчастіше використовуються виводи паралельних портів вводу/виводу, для яких ця функція є альтернативною. Джерелами запитів зовнішніх переривань також можуть бути будь-які зміни зовнішніх сигналів на деяких спеціально виділених лініях портів вводу/виводу.

Джерелами внутрішніх запитів переривань можуть служити наступні події:

- переповнення таймерів/лічильників;
- сигнали від каналів вхідного захоплення і вихідного порівняння таймерів/лічильників або від процесора подій;
- готовність пам'яті EEPROM після запису;
- сигнали переривання від додаткових модулів МК, включаючи завершення передачі або прийому інформації з одному з послідовних портів та інші.

Будь-який запит переривання надходить на обробку, якщо переривання в МК дозволені і дозволене переривання за даним запитом. Адреса, яка завантажується в програмний лічильник при переході до обробки переривання, називається "вектор переривання". У залежності від організації модуля переривань конкретного МК різні джерела переривань можуть мати різні вектори або використовувати деякі з них спільно. Використання різними перериваннями одного вектора зазвичай не викликає проблем при розробці програмного забезпечення, тому що апаратна частина МК фіксована, а контролер найчастіше виконує одну-єдину програму.

Питання про пріоритети при одночасному надходженні декількох запитів на переривання вирішується в різних МК по-різному. Є МК з однорівневою системою пріоритетів (усі запити рівноцінні), багаторівневою системою з фіксованими пріоритетами і багаторівневою програмованою системою пріоритетів.

Окремо необхідно описати апаратні переривання, пов'язані з включенням живлення, подачею сигналу ініціалізації МК і переповненням сторожового таймера. Вони мають немаскований характер і найчастіше поділяють один загальний вектор переривання. Це цілком логічно, оскільки результатом кожної з цих подій є початкова ініціалізація МК.

Наприклад, послідовний порт або UART (універсальний асинхронний приймач) мікроконтролера повинен бути обслужений при отриманні нового символу. Якщо цього не зробити до приходу наступного символу, то даний символ може бути втрачений. При надходженні нового символу UART генерує

переривання. Мікроконтролер зупиняє виконання основної програми (вашого додатку) і переходить на програму обробки переривань (ISR), призначену для даного переривання. В даному випадку це переривання за отриманим символу. Програма обробки ISR зчитує символ з UART, поміщає в буфер, потім очищає прапор переривання і виконує повернення в основну програму, продовжуючи її з точки виклику. Все це відбувається у фоновому режимі і не впливає безпосередньо на основний код роботи програми.

Якщо в процесі роботи запускається багато переривань або переривання генерує швидкодіючий таймер, основна програма буде виконуватися повільніше, так як мікроконтролер розподіляє свій машинний час між основною програмою і всіма функціями обробки переривань.

Arduino надає свої функції для роботи з зовнішніми перериваннями. Ці функції оголошені у файлі:

\ Hardware \ cores \ arduino \ wiring.h

і реалізовані в файлі:

\ Hardware \ cores \ arduino \ WInterrupts.c

Їх всього дві:

- *attachInterrupt*
- *detachInterrupt*.

attachInterrupt (interrupt, function, mode)

параметри

interrupt: номер переривання (0 – pin D2, 1 – pin D3,)

function: функція, яку необхідно викликати при виникненні переривання; ця функція повинна бути без параметрів і не повертати ніяких значень. Таку функцію іноді називають оброблювачем переривання.

mode: визначає умову, за якої має спрацьовувати переривання. Може приймати одне з чотирьох визначених значень:

LOW - переривання буде спрацьовувати щоразу, коли на виводі присутній низький рівень сигналу

CHANGE - переривання буде спрацьовувати щоразу, коли змінюється стан виведення

RISING - переривання спрацює, коли стан виведення зміниться з низького рівня на високий

FALLING - переривання спрацює, коли стан виведення зміниться з високого рівня на низький.

В Arduino Due є ще одне значення:

HIGH - переривання буде спрацьовувати щоразу, коли на виводі присутній високий рівень сигналу (тільки для Arduino Due).

Значення, що повертаються – немає

detachInterrupt (pin)

Опис: Забороняє задане переривання.

Управління перериваннями

noInterrupts ()

Параметри Ні. **Значення, що повертаються** - немає

Забороняє переривання. Для чого це робиться? Для того, щоб виключити доступ до даних в процесі виконання переривання.

interrupts ()

Параметри - Ні. **Значення, що повертаються** - немає

Опис: Повторно дозволяє переривання (після того, як вони були відключені функцією *noInterrupts ().*

Аналогове введення / виведення функція ШІМ

analogWrite (pin, value)

параметри:

pin - висновок, на якому буде формуватися напругу.

value - коефіцієнт заповнення - лежить в межах від 0 (завжди вимкнений) до 255 (завжди включений).

Значення, що повертаються: немає

Опис: Формує заданий аналогове напруга на виводі у вигляді ШІМ-сигналу. Може використовуватися для варіювання яскравості світіння світлодіода або управління швидкістю обертання двигуна. Після виклику *analogWrite* (), на виводі буде безперервно генеруватися ШІМ-сигнал з заданим коефіцієнтом заповнення до наступного виклику функції *analogWrite()* (або до моменту виклику *digitalRead ()* або *digitalWrite ()*, взаємодіючих з цим же виводом). Частота ШІМ становить приблизно 490 Гц.

На більшості плат Arduino (на базі мікроконтролерів ATmega168 або ATmega328) функція *analogWrite ()* працює з виводами 3, 5, 6, 9, 10 і 11. На Arduino Mega функція працює з виводами з 2 по 13.

Функція *analogWrite ()* не має нічого спільного з аналоговими виводами і функцією *analogRead().*

програмування АЦП

Дискретизація - це уявлення неперервної функції (тобто якогось сигналу) у вигляді ряду дискретних відліків.

При квантуванні шкала сигналу розбивається на рівні. Відлік поміщають в підготовлену сітку і перетворять в найближчий номер рівня квантування.

analogReference (type)

параметри:

type - тип джерела опорного напруги (DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56 або EXTERNAL).

Значення, що повертаються Hi.

Опис: Встановлює джерело опорного напруги, що використовується при зчитуванні аналогового сигналу (іншими словами, задає максимальне значення вхідного діапазону). Для вибору джерела опорного напруги доступні наступні значення:

DEFAULT: опорна напруга за замовчуванням, рівне 5 В (на 5В-платах Ардуіно) або 3.3 В (на 3.3В-платах Ардуіно)

INTERNAL: внутрішньо опорна напруга, рівне 1.1 В в мікроконтролерах ATmega168 і ATmega328, або 2.56 В в мікроконтролері ATmega8 (не доступно в Arduino Mega)

INTERNAL1V1: внутрішнє опорне напруга 1.1 В (тільки для Arduino Mega)

INTERNAL2V56: внутрішнє опорне напруга 2.56 В (тільки для Arduino Mega)

EXTERNAL: як опорного напруги буде використовуватися напруга, прикладена до висновку AREF (від 0 до 5В).

analogRead (pin)

параметри:

pin: номер виводу, з якого буде зчитуватися напруга (0 - 5 для більшості плат, 0 - 7 для Mini і Nano, 0 - 15 для Mega)

Значення, що повертаються: ціле число int (від 0 до 1023)

Опис: Зчитує величину напруги з зазначеного аналогового виведення. У складі Ардуіно є 6-канальний (8-канальний - в Mini і Nano, 16 - в Mega) 10-бітний аналогово-цифровий перетворювач, який перетворює вхідну напругу з діапазону 0 - 5 В в цілочисельні значення в межах від 0 до 1023 відповідно . Роздільна здатність АЦП становить: 5 В / 1024 значення або 0.0049 В (4.9 мВ) на одне значення. Вхідний діапазон і роздільна здатність можуть змінюватися за допомогою функції *analogReference ()*.

Для зчитування значення з аналогового входу потрібно близько 100 мікросекунд (0.0001 с), тому максимальна частота опитування виведення приблизно дорівнює 10 000 разів в секунду.

Якщо аналоговий вхід ні до чого не підключений, значення, що повертається функцією *analogRead (),* буде випадковим. Змінюватися під впливом декількох факторів: величина напруги на інших аналогових входах, наведення від руки поблизу плати і т.д.

Контрольні питання

1. Навіщо потрібні переривання в роботі мікроконтролера або мікропроцесора

2. Основні функції для роботи з перериваннями

3. Навести приклад коду ініціалізації та обробки переривань

4. Описати функції налаштування параметрів АЦП та ШИМ

5. Які є особливості використання та обмеження для функцій АЦП та ШИМ

Лекція 10. Клас String.

10.1. Таблиця ASCII символів

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

10.2 Клас String

Клас String з'явився в ядрі, починаючи з версії 0019. Він дозволяє здійснювати більш складну обробку текстових рядків, ніж звичайні масиви символів. За допомогою класу String можливо об'єднувати і розширювати рядки, здійснювати пошук і заміну підрядків, і багато іншого. Він займає більше пам'яті, ніж простий масив символів, але надає більше можливостей**.**

Загально прийнято називати рядки з масивів символів з маленькою літери "s" (string), а екземпляри класу String - з великою "S". Слід мати на увазі, що рядкові константи, укладені в подвійні лапки, інтерпретуються як масиви символів, а не екземпляри класу String.

String()

Опис

Створює екземпляр класу *String*. Існує кілька різних версій цієї функції, які створюють об'єкт String з різних типів даних (іншими словами, формують рядок з послідовності символів):

• строкова константа в подвійних лапках (тобто масив символів)
• одиночний символ в одинарних лапках
• другий примірник класу String
• цілочисельна константа типу int або long
• цілочисельна константа типу int або long із зазначенням підстави системи числення
• цілочисельна змінна типу int або long
• цілочисельна змінна типу int або long із зазначенням підстави системи числення

При створенні об'єкта String з числа, результуючий рядок буде містити ASCII-представлення цього числа. За замовчуванням вважається, що число зазначено в десятковій системі, тому:

```
String string = String (13)
```

приведе до створення рядка "13". Можна вказувати підставу системи числення, наприклад:

```
String string = String (13, HEX)
```

приведе до створення рядка "D", що є шістнадцяткові поданням десяткового числа 13. Те ж саме в двійковій системі:

```
String string = String (13, BIN)
```

приведе до створення рядка "1101", що є двійковим поданням числа 13.

рядки *string*

Опис: Текстові рядки можуть бути оголошені двома способами: можна використовувати тип даних String, який входить в ядро, починаючи з версії 0019; або оголосити рядок як масив символів char з нульовим символом в кінці. Далі описано другий спосіб.

Приклади:

Нижче наведені приклади правильного оголошення рядків.

```
char Str1[15];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

Допустимі операції при оголошенні рядків

• Оголосити масив символів без його ініціалізації (Str1)
• Оголосити масив символів з одним надлишковим елементом, компілятор сам додасть необхідний нульовий символ (Str2)
• Додати нульовий символ явно (Str3)
• Ініціалізувати масив за допомогою строкової константи, укладеної в лапки; компілятор створить масив необхідного розміру з нульовим символом в кінці (Str4)
• Ініціалізувати масив за допомогою строкової константи, явно вказавши його розмір (Str5)

• Ініціалізувати масив надлишкового розміру, залишивши місце для більш довгих рядків (Str6)

Нульовий завершальний символ

Як правило, всі рядки завершуються нульовим символом (ASCII код 0), який дозволяє функцій (подібним Serial.print ()) визначати довжину рядка. Без цього символу вони продовжували б послідовно зчитувати байти пам'яті, які фактично вже не були б частиною рядка.

По суті, це означає, що довжина вашої рядка повинна бути на 1 символ більше, ніж текст, який ви хотіли б у ній зберігати. Саме тому Str2 і Str5 повинні бути довжиною 8 символів, не дивлячись на те, що слово "arduino" займає всього 7 - остання позиція автоматично заповнюється нульовим символом. Розмір Str4 автоматично стане рівним 8 - один символ потрібно для завершального нуля. У рядку Str3 ми самостійно вказали нульовий символ (позначається '\ 0').

Слід мати на увазі, що в цілому можна оголосити рядок і без завершального нульового символу (наприклад, якщо задати довжину Str2 що дорівнює 7, а не 8). Однак це призведе до непрацездатності більшості строкових функцій, тому не слід навмисно так робити. Така помилка може бути причиною дивної поведінки або появи сторонніх символів при роботі з рядками.

Одинарні або подвійні лапки?

Рядки завжди оголошуються в подвійних лапках ("Abc"), а символи завжди оголошуються в одинарних лапках ('A').

String (val)
String (eval, base)
параметри

val: змінна, значення якої необхідно представити у вигляді об'єкта String - string, char, byte, int, long, unsigned int, unsigned long

base (не обов'язково параметра) - основа системи числення, в якому необхідно представити ціле число

Значення, що повертаються немає

indexOf()

Опис Здійснює пошук символу або підрядка в рядку (String). За замовчуванням, пошук здійснюється з початку рядка, однак можна вказати і певну позицію, з якою необхідно почати пошук. Така можливість дозволяє знаходити всі входження підрядка в рядку.

синтаксис
string.indexOf(val)
string.indexOf(val, from)
Параметри

string: змінна типу String
val: шукане значення - char або String
from: початкова позиція для пошуку
Значення, що повертаються Індекс подстроки val в рядку String, або -1, якщо підрядок не знайдено.

lastIndexOf ()
Опис Здійснює пошук символу або підрядка в рядку (String). За замовчуванням, пошук здійснюється з кінця рядка, однак можна вказати і певну позицію, з якої необхідно переглядати рядок у зворотному порядку. Така можливість дозволяє знаходити всі входження підрядка в рядку.

синтаксис
string.lastIndexOf (val)
string.lastIndexOf (val, from)
параметри
string: змінна типу String
val: шукане значення - char або String
from: початкова позиція для пошуку в зворотному порядку
значення, що повертаються
Індекс подстроки val в рядку String, або -1, якщо підрядок не знайдено.

length ()
Опис Повертає довжину рядка String в символах. (Зверніть увагу, що при підрахунку кількості символів функція не враховує завершальний нульовий символ).

синтаксис
string.length ()
параметри
string: змінна типу String
значення, що повертаються
Довжина рядка String в символах

substring ()
Опис
Повертає підрядок в рядку (String). Функція приймає два параметри: початковий індекс і кінцевий індекс в рядку (необов'язковий параметр). При цьому початковий індекс є включно (відповідний символ буде включений в підрядок), а кінцевий індекс - НЕ включно (відповідний символ не включається в результуючий підрядок). Якщо кінцевий індекс відсутній, то повертається функцією значення буде містити всі символи від початкового індексу і до кінця рядка.

синтаксис

string.substring (from)
string.substring (from, to)

параметри
string: змінна типу String
from: початковий індекс в рядку
to (необов'язковий параметр): кінцевий індекс в рядку

Значення, що повертаються підстрока

charAt ()
Опис
Повертає вказаний символ з рядка String.

синтаксис
string.charAt (n)
параметри
string: змінна String
n: номера символу
значення, що повертаються
n-ний символ рядка String

compareTo()
Опис Перевіряє два рядки типу String на рівність, а також дозволяє визначити, який з порівнюваних рядків йде раніше іншої в алфавітному порядку. Рядки порівнюються посимвольно по ASCII-коду кожного символу. Наприклад, символ 'a' йде до символу 'b', але після символу 'A'. Всі цифри випливають до букв.

синтаксис
string.compareTo (string2)
параметри
string: змінна типу String
string2: друга змінна типу String
значення, що повертаються

- негативне число: якщо рядок string йде до рядка string2 в алфавітному порядку
- 0: якщо рядок string еквівалентна рядку string2
- позитивне число: якщо string йде після string2

toInt ()
Опис Перетворює рядок (String) в ціле число. Рядок повинен починатися з символьного запису цілого числа. Якщо ж рядок містить не цілочисельне значення, функція припинить перетворення.

синтаксис
string.toInt ()
параметри string: змінна типу String
значення, що повертаються long

Контрольні питання

1. Призначення таблиці ASCII символів

2. Призначення та особливості використання класу String

3. Яким чином можна задати рядок? Які можна виконувати операції з рядками?

4. Яким чином можна передавати набір чисельних даних у вигляді рядка і як ці дані екстрагувати? Наведіть приклад коду.

Лекція 11. Двопроводовий послідовний інтерфейс TWI (I2C)

Двопроводовий послідовний інтерфейс TWI ідеально підходить для типових додатків на мікроконтролерах і вимагає тільки дві лінії зв'язку. Протокол TWI дозволяє проектувальнику системи зовні зв'язати до 128 різних пристроїв через одну двухпроводную двосторонню шину, де одна лінія - лінія синхронізації SCL і одна - лінія даних SDA. В якості зовнішніх апаратних компонентів, які потрібні для реалізації шини, потрібен лише підтягуючий до плюса живлення резистор на кожній лінії шини. Всі пристрої, які підключені до шини, мають свої індивідуальні адреси, а механізм визначення вмісту шини підтримується протоколом.

Відмінні особливості:

- Гнучкий, простий, ефективний послідовний комунікаційний інтерфейс, що вимагає тільки дві лінії зв'язку
- Підтримка як Master, так і Slave режимів роботи
- Можливість роботи, як приймача, так і як передавача
- 7-розр. адресний простір дозволяє підключити до шини до 128 підлеглих пристроїв
- Швидкість передачі даних до 400 кГц
- Схема шумозниження, яка підвищує стійкість до завад на лініях шини
- Програмована адреса для підлеглого режиму з підтримкою загального виклику
- Пробудження мікроконтролера з режиму сну при визначенні заданої адреси на шині

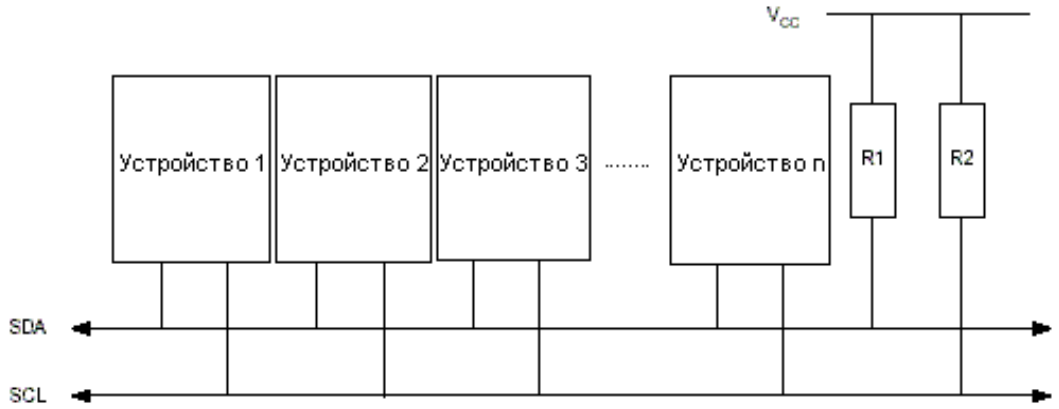


Рис. 11.1

Як показано на малюнку 11.1, обидві лінії шини підключені до позитивної шині живлення через підтягуючі резистори. У всіх сумісних з TWI пристроях як драйвер шини використовуються транзистор або з відкритим стоком або з відкритим колектором. Цим реалізована функція монтажного І, яка дуже

важлива для двобічної роботи інтерфейсу. Низький логічний рівень на лінії шини TWI генерується, якщо одне або більше з TWI-пристроїв виводить лог. 0. Високий рівень на лінії присутній, якщо все TWI-пристрої перейшли у третій високоімпедансний стан, дозволяючи підтягуючим резисторам задати рівень лог. 1. Зверніть увагу, що при підключенні до шини TWI декількох AVR-мікроконтролерів, для роботи шини повинні бути запитані всі ці мікроконтролери.

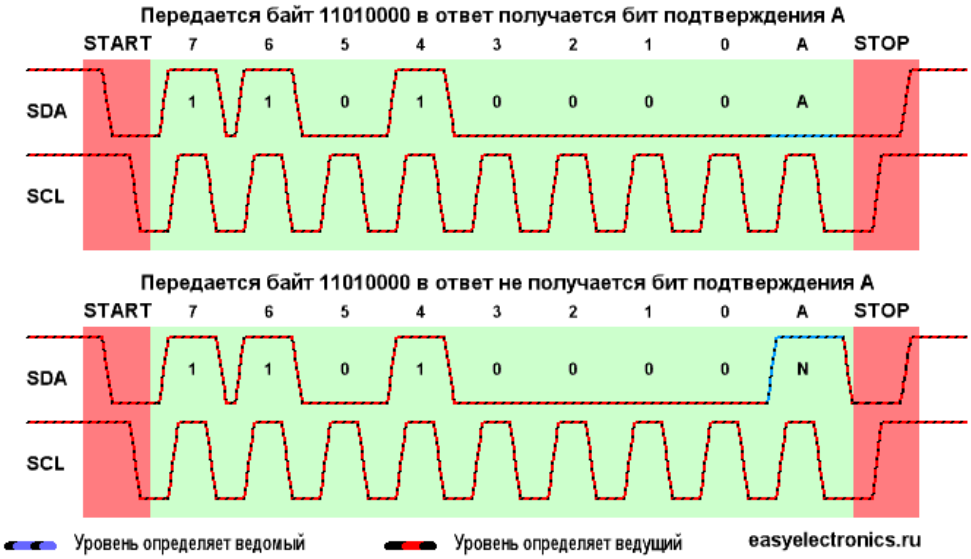
Кількість пристроїв, яку може бути підключено до однієї шині обмежується гранично допустимої ємністю шини (400 пФ) і 7-розр. адресним простором. Підтримуються два різних набору технічних вимог, де один набір для шин зі швидкістю передачі даних нижче 100 кГц і один дійсний для швидкостей понад 400 кГц.

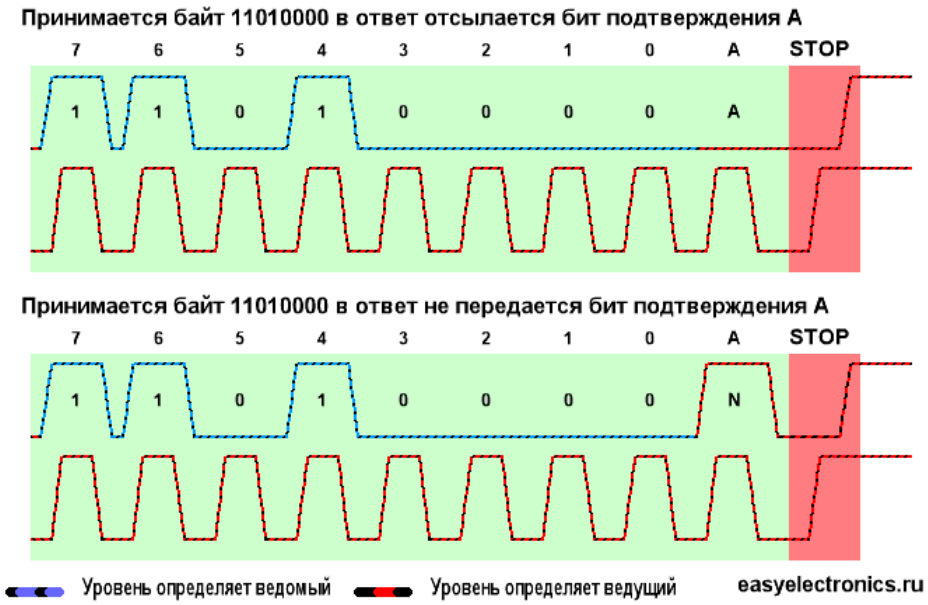
Вся передача даних складається з Стартовою посилки, бітів і стоповою посилки. Початок передачі визначається Start послідовністю - провал SDA при високому рівні SCL.

При передачі інформації від Master до Slave, провідний генерує такти на SCL і видає біти на SDA. Які ведений зчитує коли SCL стає 1.

При передачі інформації від Slave до Master, провідний генерує такти на SCL і дивиться що там ведений творить з лінією SDA - зчитує дані. А ведений, коли SCL йде в 0, виставляє на SDA біт, який майстер зчитує коли підніме SCL назад.

Закінчується все STOP послідовністю. Коли при високому рівні на SCL лінія SDA переходить з низького на високий рівень.

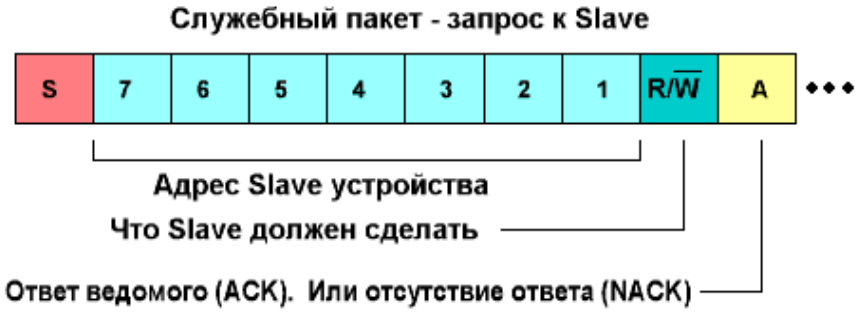




Перший пакет від ведучого до веденого - це фізична адреса пристрою і біт напрямку.

Сам адреса складається з семи біт (ось чому до 127 пристроїв на шині), а восьмий біт означає що буде робити Slave на наступному байті - приймати або передавати дані. Дев'ятим бітом йде біт підтвердження ACK. Якщо Slave розпізнав свою адресу повністю, то на дев'ятому такті він переведе лінію SDA в 0, згенерувавши ACK - тобто Зрозумів. Тоді Майстер продовжить передачу даних. Якщо Slave на відповів, тобто SDA на дев'ятому такті не буде переведено в 0 (не буде ACK), то Майстер припинить свої спроби підєднатися.

Після адресного пакета йдуть пакети з даними в ту або іншу сторону, в залежності від біта RW в заголовному пакеті.



Робота с I2C / TWI на платах Ардуіно

На платі Нано I2C: виводи 4 (SDA) і 5 (SCL).
Бібліотека Wire дозволяє Ардуіно взаємодіяти з різними пристроями по інтерфейсу I2C / TWI. Згідно з протоколом I2C, адреса пристрою може складатися як з 7, так і з 8 біт. Як правило, 7 біт ідентифікують пристрій, в той час, як восьмий біт задає напрям передачі даних: від пристрою (читання) або до

нього (запис). Всі функції бібліотеки Wire використовують 7-бітну адресацію, тим самим обмежуючи діапазон можливих адрес в межах 0 - 127.

Функції I2C / TWI на платах Ардуіно

Wire.begin()
Wire.begin(address)
Параметри: address - 7-бітову адресу відомого пристрою (необов'язковий параметр); якщо адреса не вказана, то Ардуіно виступає в ролі ведучого пристрою (master).
Значення, що повертаються - немає
Опис - Ініціалізує бібліотеку Wire і підключає Ардуіно до шини I2C в ролі ведучого (master) або веденого (slave) пристрою. Як правило, ця функція викликається тільки один раз.

Wire.requestFrom (address, quantity)
Wire.requestFrom (address, quantity, stop)
Параметри: address - 7-бітову адресу відомого пристрою, у якого запитуються дані
quantity: кількість запитуваних байт
stop: boolean. При значенні true буде відправлений запит з стоповим бітом, що дозволить звільнити шину. При значенні false - з'єднання буде підтримуватися в активному стані.
Значення, що повертаються - byte: кількість байт, повернутих веденим пристроєм
Опис: Функція запрошує дані у відомого пристрою (slave); як правило, використовується тільки провідним пристроєм (Master). Після виклику requestFrom () запитувані дані повинні бути лічені за допомогою функцій *available ()* і *read ().*
Починаючи з версії Ардуіно 1.0.1, функція *requestFrom ()* може приймати третій параметр - логічне значення, що забезпечує кращу сумісність з деякими I2C-пристроями.
Якщо цей параметр дорівнює true, то функція *requestFrom ()* відправить запит з стоповим бітом, що дозволить звільнити шину I2C.
Якщо цей параметр дорівнює false, то після відправлення запиту шина як і раніше буде зайнята, що запобігає надсиланню сторонніх повідомлень іншими провідними пристроями. Цей режим дозволяє Майстру відправляти по декілька запитів за один сеанс.
Значення за замовчуванням - true.

Wire.beginTransmission(address)
Параметри address: 7-бітову адресу пристрою
Значення, що повертаються немає
Опис Починає процедуру передачі даних по інтерфейсу I2C відомому пристрою з вказаною адресою. Для подальшої відправки даних, необхідно

спершу поставити їх в чергу за допомогою функції write (), після чого здійснити, безпосередньо, передачу функцією endTransmission ().

Wire.endTransmission()
Wire.endTransmission(stop)
Параметри stop: boolean. При значенні true буде відправлений запит з стоповим бітом, що дозволить звільнити шину. При значенні false - з'єднання буде підтримуватися в активному стані, що запобігає надсиланню сторонніх повідомлень іншими провідними пристроями. Цей режим дозволяє Майстру відправляти по декілька запитів за один сеанс.
Значення, що повертаються byte, байт даних, що характеризує статус передачі:
0: передача успішна
1: обсяг даних занадто великий для буфера передачі
2: отриманий NACK при передачі адреси
3: отриманий NACK при передачі даних
4: інша помилка
Опис Завершує процедуру передачі даних відомому пристрою, ініційовану функцією beginTransmission (). При цьому функція відправляє байти, поставлені в чергу функцією write ().
Значення за замовчуванням - true.

Wire.write(value)
Wire.write(string)
Wire.write(data, length)
Параметри value: значення, яке необхідно відправити у вигляді одиночного байта
string: рядок, яку необхідно відправити у вигляді послідовності байт
data: масив даних, який необхідно відправити у вигляді декількох байт
length: кількість переданих байт
Значення, що повертаються byte
Опис Функція write () повертає кількість записаних байт. Зчитування цього значення не обов'язково

Wire.available ()
Параметри немає
Значення, що повертаються Кількість байт, доступних для зчитування.
Опис Повертає кількість байт, доступних для зчитування функцією read (). На провідному пристрої (Master), ця функція має викликатися після функції requestFrom (), а на відомому (Slave) - всередині обробника onReceive (). Функція available () є спадкоємцем допоміжного класу Stream.

Wire.read ()
Параметри немає

Значення, що повертаються Черговий отриманий байт
Опис Ця функція зчитує байт даних, отриманий провідним пристроєм від веденого (або навпаки) в результаті виконання функції requestFrom (). Функція read () є спадкоємцем допоміжного класу Stream.

Wire.onReceive (handler)
Параметри handler: функція, яку необхідно викликати при надходженні даних від провідного пристрою; ця функція не повинна повертати ніяких значень і може приймати тільки один параметр (int), що описує кількість надійшли байт. Наприклад: void myHandler (int numBytes)**Значення, що повертаються** немає
Опис На відомому пристрої дозволяє призначити функцію, яка буде автоматично викликатися при надходженні даних від Майстра.

Wire.on Request (handler)
Параметри handler: викликається, без параметрів, не повинна повертати ніяких значень. Наприклад: void myHandler ()
Значення, що повертаються немає
Опис На відомому пристрої дозволяє призначити функцію, яка буде автоматично викликатися при отримання запиту від Майстра.

Протокол I2C / TWI на платах Ардуіно – модуль DS3231

DS3231 - високоточні годинники реального часу (RTC) з вбудованими I2C інтерфейсом, термокомпенсованим кварцовим генератором (TCXO) і кварцовим резонатором. Прилад має вхід для підключення резервного автономного джерела живлення, що дозволяє здійснювати хронометраж і вимір температури навіть при відключеній основній напрузі живлення. Вбудований кварцовий резонатор підвищує термін служби приладу і зменшує необхідну кількість зовнішніх елементів. DS3231 доступний в модифікаціях з комерційно і індустріальним робочим температурним діапазоном і упакований в 300 mil 16 контактний SO корпус.

RTC забезпечує відлік секунд, хвилин, годин, днів тижня, днів місяця і року. Дата кінця місяця визначається автоматично з урахуванням високосного року. Годинник реального часу працюють в 24 або 12- годинному форматі з індикацією поточної половини доби (AM / PM). Прилад має два щоденних будильника і вихід прямокутного сигналу з програмованої частотою. Обмін даними з приладом ведеться через вбудований послідовний I2C сумісний інтерфейс.

Прецизійне термокомпенсоване джерело опорної напруги і схема порівняння відстежують напругу основного живлення VCC і при його зниженні нижче заданого порогу формують сигнал скидання і здійснюють перемикання

схеми на роботу від резервного джерела живлення. Додатковий вивід RST може використовуватися для зовнішнього скидання.

Опис регістрів RTC DS3231

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/DT	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

Будильники

DS3231 містить два будильники час-дня / дати. Будильник 1 може бути встановлений шляхом запису в регістри 07h - 0Ah. Будильник 2 може бути встановлений шляхом запису в регістри 0Bh до 0Dh.

Сигнали можуть бути запрограмовані (при включенні будильника і INTCN біта регістра управління), щоб активувати INT / SQW вихід при настанні умові включення сигналу будильника. Біти 7 кожного час-дня / дати регістри є бітами маски (таблиця 2). Коли все біти маски для кожного будильника є логічним 0, сигнал тривоги відбувається тільки тоді, коли значення в регістрах часу відповідають відповідним значенням, збереженим в регістрах час-дня / дати. Будильники також можуть бути запрограмовані, щоб повторити сигнал кожну секунду, хвилина, годину, день або дату. У таблиці 2 наведено можливі налаштування. Конфігурації, які не перераховані в таблиці призведуть до нелогічної операції.

Біти DY / DT (біт 6 регістрів будильника день / дата) контролюють чи має значення будильника, яке зберігається в бітах від 0 до 5 цього регістру відображати день тижня або дату місяця. Якщо DY / DT записується в логічний 0, сигнал будильники буде у результаті співпадіння з датою місяця. Якщо DY / DT має значення 1, то сигнал тривоги буде у результаті співпадіння з днем тижня. Коли значення регістра RTC збігаються зі значеннями регістра будильника, відповідний прапор будильника 'A1F' або біт 'A2F' встановлюється

в 1. Якщо відповідний сигнал переривання будильника 'A1IE' або 'A2IE' також встановлюється в 1 і біт INTCN встановлюється в 1, то будильник буде активувати INT / SQW сигнал. Співпадіння перевіряється кожну секунду.

DY/$\overline{DT}$	ALARM 1 REGISTER MASK BITS (BIT 7)				ALARM RATE
	A1M4	A1M3	A1M2	A1M1	
X	1	1	1	1	Alarm once per second
X	1	1	1	0	Alarm when seconds match
X	1	1	0	0	Alarm when minutes and seconds match
X	1	0	0	0	Alarm when hours, minutes, and seconds match
0	0	0	0	0	Alarm when date, hours, minutes, and seconds match
1	0	0	0	0	Alarm when day, hours, minutes, and seconds match

DY/$\overline{DT}$	ALARM 2 REGISTER MASK BITS (BIT 7)			ALARM RATE
	A2M4	A2M3	A2M2	
X	1	1	1	Alarm once per minute (00 seconds of every minute)
X	1	1	0	Alarm when minutes match
X	1	0	0	Alarm when hours and minutes match
0	0	0	0	Alarm when date, hours, and minutes match
1	0	0	0	Alarm when day, hours, and minutes match

Зчитування температури.

Temperature Register (Upper Byte) (11h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	Sign	Data	Data	Data	Data	Data	Data	Data
POR:	0	0	0	0	0	0	0	0

Temperature Register (Lower Byte) (12h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	Data	Data	0	0	0	0	0	0
POR:	0	0	0	0	0	0	0	0

Температура представлена у вигляді 10-бітового коду з роздільною здатністю 0.25 ° C і доступна за адресами 11h та 12h. Температура кодується у двох додаткових форматах. Верхні 8 біт, цільна частина, знаходяться за адресою 11h, а нижні 2 біти, фракційна частина, знаходяться в старших байтах за адресою 12h. Наприклад, 00011001 01b = + 25,25 ° C. Після скидання живлення регістри встановлюють температуру за замовчуванням 0 ° C, а контролер починає перетворення температури. Температура читається при первинному підключенні VCC або I^2C доступу на VBAT і один раз на 64 секунди після цього. Температурні регістри оновлюються після кожної ініційованої користувачем конверсії та кожної 64-секундної конверсії. Регістри температури доступні лише для читання.

Контрольні питання

1. Призначення і особливості інтерфейсу I^2C.
2. Електричне підключення та протокол обміну даними інтерфейсу I^2C.
3. Опис функцій Arduino для обміну даними через інтерфейс I^2C
4. Опис модуля годинника DS3231 і його програмування через інтерфейс I^2C

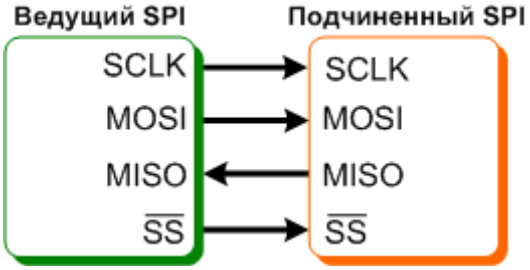
Лекція 12. Інтерфейс SPI на платах Ардуіно

SPI - популярний інтерфейс для послідовного обміну даними між мікросхемами. Інтерфейс SPI, поряд з I^2C, відноситься до найбільш широко-використовуваних інтерфейсів для з'єднання мікросхем. Спочатку він був придуманий компанією Motorola, а в даний час використовується в продукції багатьох виробників. Його найменування є абревіатурою від "Serial Peripheral Interface", що відображає його призначення - шина для підключення зовнішніх пристроїв. Шина SPI організована за принципом ведучий-підлеглий. В якості ведучого шини зазвичай виступає мікроконтролер, але їм також може бути програмована логіка, DSP-контролер або спеціалізована IC. Підключення до ведучої шини зовнішні пристрої утворюють підлеглі шини. В їх ролі виступають різного роду мікросхеми, в т.ч. запам'ятовуючі пристрої (EEPROM, Flash-пам'ять, SRAM), годинник реального часу (RTC), АЦП / ЦАП, цифрові потенціометри, спеціалізовані контролери та ін.

Головним складовим блоком інтерфейсу SPI є звичайний зсувний регістр, сигнали синхронізації і введення / виводу бітового потоку які і утворюють інтерфейсні сигнали. Таким чином, протокол SPI правильніше назвати не протокол передачі даних, а протоколом обміну даними між двома зсувними регістрами, кожен з яких одночасно виконує і функцію приймача, і функцію передавача. Неодмінною умовою передачі даних по шині SPI є генерація сигналу синхронізації шини. Цей сигнал має право генерувати тільки ведучий шини і від цього сигналу повністю залежить робота підлеглого шини.

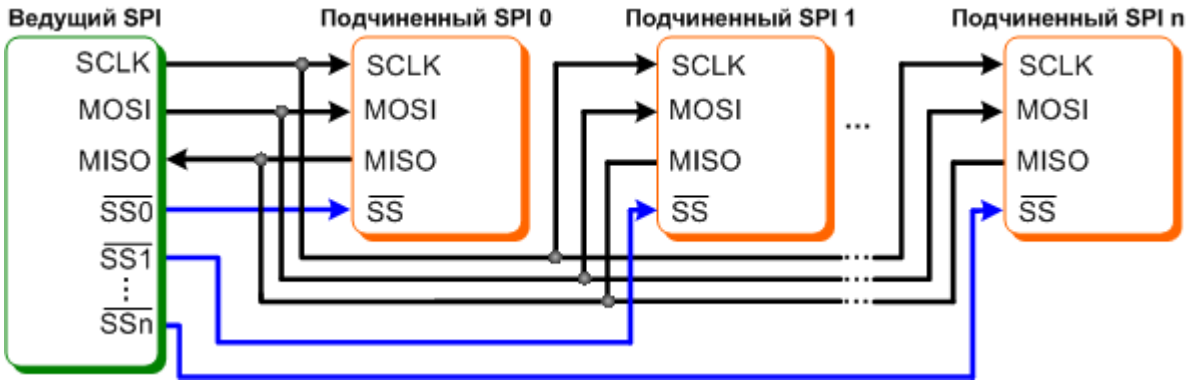
Електричне підключення

Існує три типи підключення до шини SPI, в кожному з яких беруть участь чотири сигнали. Найпростіше підключення, в якому беруть участь тільки дві мікросхеми, показано на малюнку 1. Тут, провідний шини передає дані по лінії MOSI синхронно зі згенерованим їм же сигналом SCLK, а підлеглий захоплює передані біти даних за певними напрямами прийнятого сигналу синхронізації. Одночасно з цим підлеглий відправляє свою посилку даних. Представлену схему можна спростити виключенням лінії MISO, якщо використовувана підпорядкована IC не передбачає відповідну передачу даних або в ній немає потреби. Односторонню передачу даних можна зустріти у таких мікросхем як ЦАП, цифрові потенціометри, програмовані підсилювачі і драйвери. Таким чином, розглянутий варіант підключення підпорядкованої IC вимагає 3 або 4 лінії зв'язку. Щоб підпорядкована IC приймала і передавала дані, крім наявності сигналу синхронізації, необхідно також, щоб лінія SS була переведена в низький стан. В іншому випадку, підпорядкована IC буде неактивна. Коли використовується тільки одна зовнішня IC, може виникнути спокуса виключення і лінії SS за рахунок жорсткої установки низького рівня на вході вибору підпорядкованої мікросхеми. Таке рішення вкрай небажано і може привести до збоїв або взагалі неможливості передачі даних, тому що вхід вибору мікросхеми служить для перекладу IC в її початковий стан і іноді ініціює виведення першого біта даних.

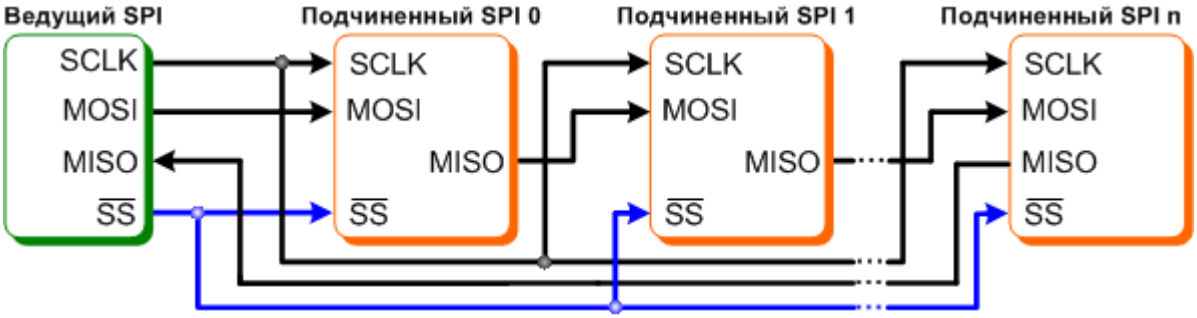


Мал. 1. Найпростіше підключення до шини SPI

При необхідності підключення до шини SPI декількох мікросхем використовується або незалежне (паралельне) підключення (рис. 2), або каскадне (послідовне) (рис. 3). Незалежне підключення більш поширене, тому що досягається при використанні будь-яких SPI-сумісних мікросхем. Тут, всі сигнали, крім вибору мікросхем, з'єднані паралельно, а ведучий шини, перекладом того чи іншого сигналу SS в низький стан, задає, з якою підпорядкованої IC він буде обмінюватися даними. Головним недоліком такого підключення є необхідність в додаткових лініях для адресації підлеглих мікросхем (загальне число ліній зв'язку дорівнює 3 + n, де n-кількість підлеглих мікросхем).



Мал. 2. Незалежне підключення до шини SPI



Мал. 3. Каскадне підключення до шини SPI

Каскадне включення вільне від цього недоліку, тому що тут з кількох мікросхем утворюється один великий зсувний регістр. Для цього вихід передачі даних однієї IC з'єднується зі входом прийому даних інший, як показано на малюнку 3. Входи вибору мікросхем тут з'єднані паралельно і, таким чином, загальне число ліній зв'язку збережено рівним 4. Однак використання каскадного підключення можливо тільки в тому випадку, якщо його підтримка

вказана в документації на використовувані мікросхеми. Щоб з'ясувати це, важливо знати, що таке підключення по-англійськи називається 'daisy-chaining'.

Протокол передачі

Протокол передачі по інтерфейсу SPI гранично простий і, по суті, ідентичний логіці роботи зсувного регістру, яка полягає у виконанні операції зсуву і, відповідно, побітного введення і виведення даних за певними фронтах сигналу синхронізації. Установка даних при передачі і вибірка при прийомі завжди виконуються по протилежних фронтах синхронізації. Це необхідно для гарантування вибірки даних після надійного їх встановлення. Якщо до цього врахувати, що в якості першого фронту в циклі передачі може виступати наростаючий або падаючий фронт, то все можливо чотири варіанти логіки роботи інтерфейсу SPI. Ці варіанти отримали назву режимів SPI і описуються двома параметрами:

CPOL - вихідний рівень сигналу синхронізації (якщо CPOL = 0, то лінія синхронізації до початку циклу передачі і після його закінчення має низький рівень (тобто перший фронт наростаючий, а останній - падаючий), інакше, якщо CPOL = 1, - високий (тобто перший фронт падаючий, а останній - наростаючий));

CPHA - фаза синхронізації; від цього параметра залежить, в якій послідовності виконується установка і вибірка даних (якщо CPHA = 0, то по передньому фронту в циклі синхронізації буде виконуватися вибірка даних, а потім, по задньому фронту, - установка даних; якщо ж CPHA = 1, то установка даних буде виконуватися по передньому фронту в циклі синхронізації, а вибірка - по задньому). Інформація по режимам SPI узагальнена в таблиці 2.

Провідна і підпорядкована мікросхеми, що працюють в різних режимах SPI, є несумісними, тому, перед вибором підлеглих мікросхем важливо уточнити, які режими підтримуються провідним шини. Апаратні модулі SPI, інтегровані в мікроконтролери, в більшості випадків підтримують можливість вибору будь-якого режиму SPI і, тому, до них можливе підключення будь-яких підлеглих SPI-мікросхем (відноситься тільки до незалежного варіанту підключення). Крім того, протокол SPI в будь-якому з режимів легко реалізується програмно.

Порівняння шини SPI з шиною I2C

Переваги шини SPI	Переваги шини I2C
Простота протоколу передачі на фізичному рівні обумовлює високу надійність і швидкодію передачі. Граничне швидкодія шини SPI вимірюється десятками мегагерц і, тому, вона ідеальна для потокової передачі великих обсягів даних і широко використовується в високошвидкісних ЦАП / АЦП, драйвери світлодіодних дисплеїв і мікросхемах пам'яті	Шина I2C залишається двухпроводной, незалежно від кількості підключеної до неї мікросхем.
Всі лінії шини SPI є односпрямованим, що істотно спрощує рішення задачі перетворення рівнів і гальванічної ізоляції мікросхем	Можливість мультімастерний роботи, коли до шини підключено кілька провідних мікросхем.
Простота програмної реалізації протоколу SPI.	Протокол I2C є більш стандартизованим, тому, користувач I2C-мікросхем більш захищений від проблем несумісності обраних компонентів.

Функції SPI в Arduino

SPI.begin ()
SPI.begin (slaveSelectPin) (тільки для Arduino Due)

Опис Функція ініціалізує роботу шини SPI, а саме: конфігурує виводи SCK, MOSI і SS як виходи, формує низький рівень сигналу на виводах SCK і MOSI, і високий рівень - на виводі SS.
Параметри slaveSelectPin: вивід SS веденого пристрою (slave) - (тільки для Arduino Due)
Зачення, що повертаються - немає

SPI.end ()
SPI.end (slaveSelectPin) (тільки для Arduino Due)
Опис Функція завершує роботу шини SPI (режим роботи виводів SPI залишається колишнім).
Параметри slaveSelectPin: висновок SS веденого пристрою (slave) - (тільки для Arduino Due)
Значення, що повертаються – немає

SPI.setBitOrder (order)
SPI.setBitOrder (slaveSelectPin, order) (тільки для Arduino Due)
Опис Функція визначає порядок проходження біт даних по шині SPI: молодшим бітом вперед (LSBFIRST) або старшим бітом вперед (MSBFIRST).
Параметри - order: одне з двох значень - LSBFIRST або MSBFIRST.

Значення, що повертаються – немає

SPI.setClockDivider (divider)
SPI.setClockDivider (slaveSelectPin, divider) (тільки для Arduino Due)
Опис Дозволяє задати тактову частоту SPI, вказавши коефіцієнт ділення тактової частоти контролера. У Ардуіно на базі AVR-мікроконтролерів можна використовувати один з наступних коефіцієнтів ділення: 2, 4, 8, 16, 32, 64 або 128. За замовчуванням тактова частота SPI в чотири рази менше тактової частоти контролера (SPI_CLOCK_DIV4). Тобто, якщо тактова частота контролера 16 МГц, то SPI буде працювати на частоті 4 МГц.

Arduino Due
В Arduino Due системну частоту можна ділити на будь-яке число в діапазоні від 1 до 255. За умовчанням встановлено коефіцієнт 21, щоб частота SPI була рівною 4 МГц, як і на інших моделях Ардуіно.
Розширені можливості в Arduino Due
Якщо при виконанні функції setClockDivider () ви вкажете один з висновків SS Arduino Due, то зазначена вами частота буде задана тільки для того пристрою на шині SPI, яке пов'язане з ці висновком.
Параметри
divider:
SPI_CLOCK_DIV2
SPI_CLOCK_DIV4
SPI_CLOCK_DIV8
SPI_CLOCK_DIV16
SPI_CLOCK_DIV32
SPI_CLOCK_DIV64
SPI_CLOCK_DIV128
(Для AVR-пристроїв)
slaveSelectPin: висновок SS (Тільки для Arduino Due)
divider: число від 1 до 255 (Тільки для Arduino Due)
Значення, що повертаються – немає

SPI.transfer (val)
SPI.transfer (slaveSelectPin, val) (тільки для Arduino Due)
SPI.transfer (slaveSelectPin, val, transferMode) (тільки для Arduino Due)
опис Здійснює передачу байта даних по шині SPI, одночасно приймаючи входить байт.
Параметри
val:
байт даних, який необхідно відправити SPI
slaveSelectPin: висновок SS (тільки Arduino Due)
transferMode:
SPI_CONTINUE: залишає висновок SS в низькому рівні, що дозволяє продовжити передачу байтів.

SPI_LAST: значення за замовчуванням - після передачі одного байта даних, висновок SS повертається в високий рівень.
Значення, що повертаються - байт даних, отриманий по шині SPI

Контрольні питання

1. Призначення і особливості інтерфейсу SPI.

2. Електричне підключення та протокол обміну даними інтерфейсу SPI.

3. Опис функцій Arduino для обміну даними через інтерфейс SPI

4. Навести переваги та недоліки інтерфейсу SPI в порівнянні з інтерфейсом I2C

Лекція 13. Інтерфейс 1-Wire

Інтерфейс 1-Wire був запропонований фірмою Dallas Semiconductor в кінці 90-х років минулого століття. Системи 1-Wire привабливі завдяки легкості монтажу, низької вартості пристроїв, можливості вибирати користувача при підключенні до функціонуючої мережі, великому числу пристроїв в мережі і т.д.

Типова система 1-Wire складається з керуючого контролера (майстра або ведучого) і одного або декількох пристроїв (ведених), приєднаних до загальної шині (рис.1)

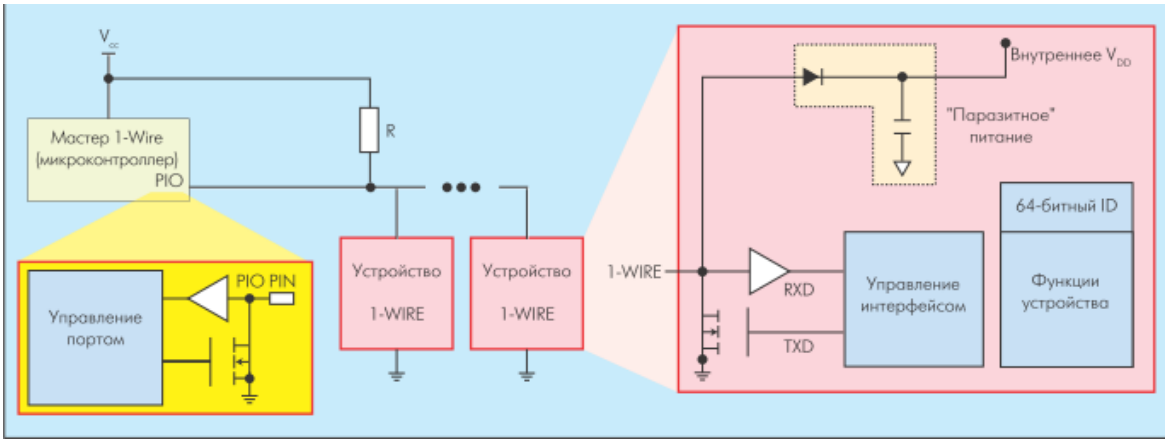


рис.1. Конфігурація системи 1-Wire

Пристрої підключаються до шини за схемою з відкритим стоком і підтягуючим резистором (див. Рис.1). Рівень сигналів в шині - від 3 до 5 В. У пасивному стані в лінії підтримується високий рівень напруги. Всі сигнали формуються за допомогою замикання сигнальної шини на землю (низький рівень напруги).

Головна особливість шини 1-Wire в тому, що вона використовує лише два дроти, один - сигнальний, інший - для заземлення пристроїв. По сигнальному проводу можливо і електроживлення пристроїв 1-Wire - так зване паразитное живлення. Джерелом живлення служить конденсатор, який заряджається від сигнальної лінії, що входить до складу відомих пристроїв кола. (див. Рис.1).

Більшість пристроїв 1-Wire підтримують дві швидкості передачі даних: стандартну - близько 15 кбіт / с і підвищену (overdrive) - близько 111 кбіт / с. Зрозуміло, що чим вище швидкість, тим більше обмежень на довжину шини і число підключаються до неї пристроїв.

Режим передачі даних по шині 1-Wire - напівдуплексний: майстер і ведені пристрої передають дані по черзі. Кожна транзакція через інтерфейс 1-Wire починається з того, що майстер передає імпульс Reset. Для цього він переводить напруга в шині на низький рівень і утримує його в цьому стані протягом 480 мкс (рис.2). Потім майстер відпускає шину, і підтягуючий резистор повертає напругу до високого логічного рівня. Всі ведені пристрої, виявивши сигнал Reset і дочекавшись його закінчення, передають свій сигнал - Presence. Він являє собою сигнал низького рівня тривалістю 100-200 мс.

Пристрій може генерувати сигнал Presence і без імпульсу Reset - наприклад, у такий спосіб воно повідомляє про себе при підключенні до шини.

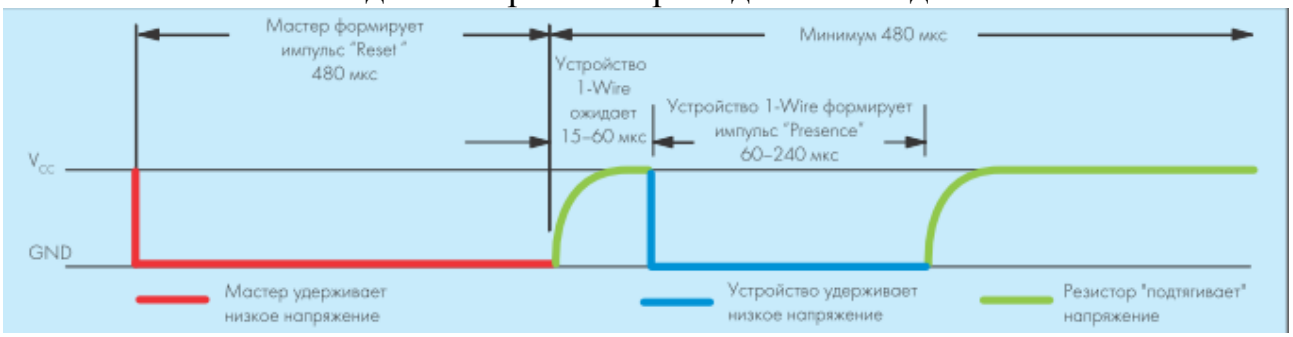


рис.2. Послідовність ініціалізації шини 1-Wire

Після передачі імпульсу Presence пристрій 1-Wire готове до прийому команд. Весь інформаційний обмін в шині відбувається під керуванням майстра. Для передачі кожного біта виділяється спеціальний часовий проміжок (таймслот) тривалістю близько 80 мкс. На початку кожного таймслота майстер переводить лінію на нульовий рівень. Якщо далі майстер хоче передати 0, він утримує напругу на низькому рівні як мінімум 60 мкс (рiс.3). При передачі одиниці майстер утримує нульове напруга 5-6 мкс, а потім відпускає лінію і вичікує приблизно 60 мкс до початку формування наступного таймслота.

Якщо майстер очікує дані від ведених пристроїв, він також позначає початок таймслота, обнулити лінію на 5-6 мкс, після чого перестає утримувати низьку напругу і протягом короткого часу слухає лінію (рiс.3б). Якщо пристрій хоче передати нуль, воно саме обнуляє лінію відразу після реєстрації імпульсу початку тайм-слота. Якщо пристрою потрібно передати одиницю, воно ніяких дій не робить. Відзначимо, що наведені значення тимчасових інтервалів відповідають стандартній швидкості передачі даних через інтерфейс 1-Wire. У режимі overdrive ці інтервали відповідно зменшуються.

Весь обмін на шині 1-Wire відбувається за допомогою спеціальних команд. Їх число для кожного типу пристроїв по-різному. Але є і мінімальний набір стандартних команд, які підтримують всі 1-Wire-пристрої - так звані ROM-команди.

Формат команд простий - ідентифікатор команди (1 байт), за яким можуть слідувати дані (ідентифікатор пристрою, корисні дані і т.п.). Всі пристрої в мережі знають довжину кожної команди.

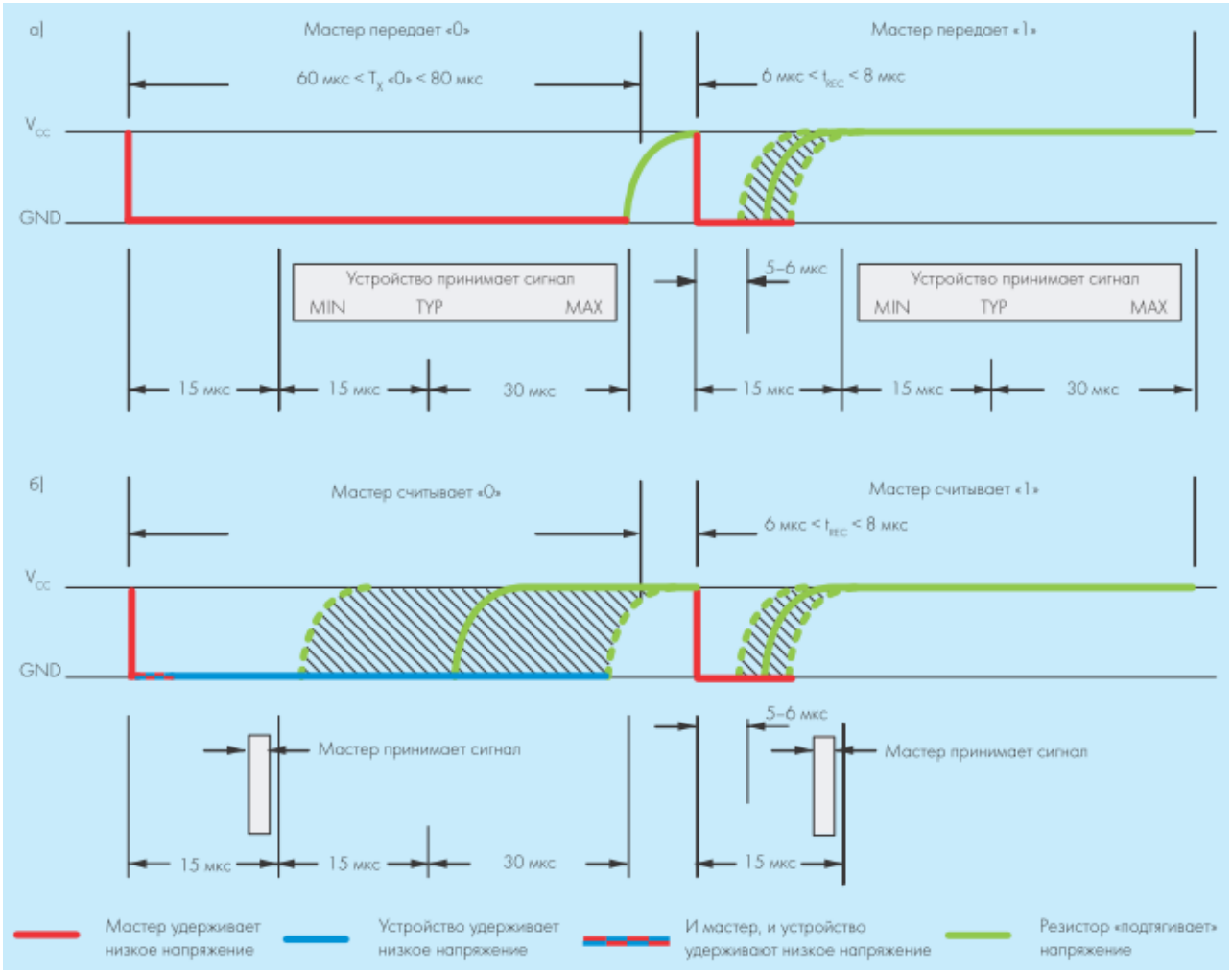


рис.3. Передача інформаційних бітів по шині 1-Wire: а - майстер передає сигнали, б - майстер зчитує сигнали

У кожного пристрою 1-Wire є 64-розрядний ідентифікатор (ID). Він складається з 8-розрядного коду сімейства, який ідентифікує тип пристрою і та функції, які він підтримує, 48-розрядної серійного номера і 8-бітного поля коду циклічного надлишкового контролю (CRC-8). ID вводиться при виготовленні пристрою і зберігається в ПЗУ. Фірма Maxim гарантує, що один раз використаний адреса ніколи не повториться в іншому пристрої. Справді, 48 біт - це $2{,}81 \cdot 10^{14}$ різних чисел. Якщо виробляти 1000 млрд (10^{12}) різних пристроїв щорічно, то все серійні номери можна використовувати не раніше ніж через 281 рік - і це тільки для одного сімейства.

Весь обмін командами ініціює майстер. Початок нового циклу транзакцій він зазначає командою Reset, і, отримавши підтвердження, вибирає пристрій спеціальної командою MATCH ROM, передаючи її ідентифікатор (55_{16}) і 64 біта ID адресується пристрою. Отримавши таку команду, ведений пристрій з даним ID очікує нових команд від майстра, а всі інші залишаються в пасивному стані до наступної команди Reset. В системі з одним пристроєм можна не передавати ID, використовуючи команду SKIP ROM. В результаті ведене пристрій вважає себе обраним без отримання адреси.

Після того, як майстер вибрав пристрій для взаємодії, можна починати процес управління цим пристроєм і обміну даними з ним. Для цього використовуються команди, які специфічні для кожного типу пристроїв.

Але щоб почати роботу з певним пристроєм, майстер повинен знати його ID. Якщо в системі тільки одне відоме пристрій, його адреса можна визначити за допомогою команди READ ROM. У відповідь на команду READ ROM пристрій передає свій 64-бітову адресу (рис.4) .

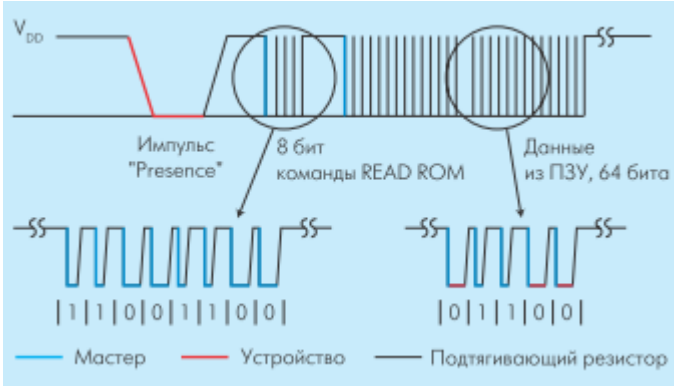


рис.4. Читання адреси пристрою

Якщо ж в системі декілька пристроїв з невідомими ID, спроба використовувати команду READ ROM призводить до колізії. У цьому випадку для визначення адрес використовується спеціальний алгоритм пошуку, в основі якого лежить команда SEARCH ROM. Майстер передає команду SEARCH ROM. У відповідь всі пристрої, підключені до шини, висилають молодший біт своєї адреси. Властивості шини 1-Wire такі, що при одночасній передачі сигналів усіма пристроями результат буде дорівнює логічному І значень всіх поселеннях бітів. Отже, сумарний відгук дорівнює 1, тільки коли сигнали від всіх пристроїв рівні 1. Після прийому першого біта адреси майстер ініціює наступний таймслот, в якому пристрій передає інвертований перший біт. Зіставляючи значення результатів запиту істинного і інверсного бітів, можна отримати певну інформацію про значеннях перших бітів адрес пристроїв (див. Таблицю).

Істинний біт	Інверсний біт	інформація
0	**0**	У поточному биті адреси є як 0, так і 1. Це так зване "розбіжність" (discrepancy)
0	**1**	У биті адреси присутні тільки нулі.
1	**0**	У биті адреси присутні тільки одиниці
1	**1**	У пошуку не бере жоден пристрій

Таким чином, при комбінаціях 0 1 і 1 0 майстер знає значення першого біта адреси, фіксує його і по тій же схемі може переходити до визначення наступного. Після отримання інверсного біта майстер передає певний біт веденим пристроям. Якщо його значення збігається зі значенням поточного біта

з адреси пристрою, то пристрій продовжує брати участь в пошуку і видає у відповідь наступний біт своєї адреси. Якщо не було "розбіжності", то значення виставляється майстром біта визначено. У разі розбіжності майстер посилає нульовий біт. Така послідовність - читання біта адреси та інверсного біта, передача біта майстром - повторюється для наступних 63 бітів адреси. Таким чином, алгоритм пошуку послідовно виключає всі пристрої, поки не залишається одне останнє - його адресу і визначається в першому циклі пошуку.

Після того, як адресу першого пристрою визначено, пошук триває для наступного пристрою. Алгоритм запам'ятовує місце останнього розбіжності і вибирає іншу гілку дерева пошуку (майстер посилає в цьому місці біт з іншим значенням). Процес триває до тих пір, поки не буде пройдена гілка, відповідна останньому пристрою. В результаті пошуку стають відомі адреси всіх пристроїв, під'єднаних до шини, і їх число.

Можливість ідентифікації і швидкого включення в мережу тільки що підключеного пристрою робить 1-Wire ефективним рішенням для багатьох додатків. На практиці це означає, що прилад досить просто підключити до мережі, і всі подальші транзакції відбудуться автоматично. Наприклад, так можна вважати дані з пам'яті датчика, прочитати код електронної мітки або електронного ключа, прийняти масив значень від приладової мережі і т.п.

Не менш важливо, що мережа 1-Wire відноситься до самосинхронізується, тобто не вимагає окремої лінії для передачі тактових сигналів. І, звичайно, величезне число ID пристроїв, що підключаються вигідно виділяє її на тлі інших послідовних мереж.

Деякі застосування 1-Wire

Наявність унікальних 64-бітних адрес дозволяє широко використовувати пристрої 1-Wire в системах аутентифікації. Тут вони часто застосовуються в пристроях iБаттон. Це мікросхема з введенням на етапі виробництва 64-бітних адресою, укладена в круглий корпус з нержавіючої сталі діаметром 16 мм (MicroCAN). Такі пристрої функціонують, наприклад, в домофонних ключах (рис.7).

Мікросхеми з підтримкою 1-Wire (наприклад, DS2401, DS2431, DS28E01-100) використовуються також для ідентифікації картриджів принтерів, медичних сенсорів, ємностей з реагентами та ін. Перевага мікросхем 1-Wire в тому, що для контролю ідентифікованого пристрою потрібен всього один контакт. Такі мікросхеми укладені в спеціальний плоский корпус (SFN - Single Flat No lead) розміром 6 * 6 мм, який полегшує їх приєднання до пристрою.

Ще одна поширена застосування 1-Wire - системи автоматизації. В першу чергу це системи багатоточкового вимірювання температури різних середовищ і моніторингу теплового режиму приміщень. Температуру можна вимірювати датчиками виробництва тієї ж Maxim / Dallas. Найбільш популярний з них - цифровий термометр DS18S20. Він має роздільну здатність 9 біт і вимірює температуру в діапазоні від -55 до 125 ° С. Точність вимірювань складає 0,5 ° С в діапазоні -10 ... 85 ° С. Оскільки кожен термометр, як і будь-який пристрій 1-

Wire, має унікальний 64-бітову адресу, до однієї шині 1-Wire можна підключати безліч таких приладів.

Опис бібліотеки 1-Wire від PaulStoffregen (https://github.com/PaulStoffregen/OneWire)

```
// Constructor. Select pin for 1-Wire
OneWire::OneWire(uint8_t pin)

// Perform the onewire reset function.  We will wait up to 250uS for
// the bus to come high, if it doesn't then it is broken or shorted
// and we return a 0;
// Returns 1 if a device asserted a presence pulse, 0 otherwise.
//
uint8_t OneWire::reset(void)

// Write a byte. The writing code uses the active drivers to raise the
// pin high, if you need power after the write (e.g. DS18S20 in
// parasite power mode) then set 'power' to 1, otherwise the pin will
// go tri-state at the end of the write to avoid heating in a short or
// other mishap.
//
OneWire::write_bytes(const uint8_t *buf, uint16_t count, bool
power /* = 0 */)

// Read a byte
OneWire::read_bytes(uint8_t *buf, uint16_t count)

// Perform the 1-Wire Search Algorithm on the 1-Wire bus using the existing
// search state.
// Return TRUE  : device found, ROM number in ROM_NO buffer
//        FALSE : device not found, end of search
//
uint8_t OneWire::search(uint8_t *newAddr, bool search_mode /* =
true */)
```

Контрольні питання

1. Призначення і особливості інтерфейсу 1-Wire.
2. Електричне підключення та протокол обміну даними інтерфейсу 1-Wire.
3. Опис функцій Arduino для обміну даними через інтерфейс 1-Wire
4. Навести приклади застосування інтерфейсу 1-Wire

Лекція 14. Основні напрями, завдання та алгоритми ЦОС

В даний час виділяють наступні основні напрямки ЦОС (табл. 1.1):

- лінійна фільтрація;
- спектральний аналіз;
- частотно-часовий аналіз;
- адаптивна фільтрація;
- нелінійна обробка;
- багатошвидкісна обробка.

Таблиця 1.1. Основні завдання ЦОС

п/п	Напрямок	Приклади завдань
	Лінійна фільтрація	Селекція сигналу в частотній області; синтез фільтрів, узгоджених з сигналами; частотне розділення каналів; цифрові перетворювачі Гільберта і диференціатори; коректори характеристик каналів
	Спектральний аналіз	Обробка мовних, звукових, сейсмічних, гідроакустичних сигналів; розпізнавання образів
	Частотно-часовий аналіз	Компресія зображень, гідро- і радіолокація, різноманітні завдання виявлення
	Адаптивна фільтрація	Обробка мови, зображень, розпізнавання образів, придушення шумів, адаптивні антенні решітки
	Нелінійна обробка	Обчислення кореляцій, медіанна фільтрація; синтез амплітудних, фазових, частотних детекторів, обробка мови, векторне кодування
	Багатошвидкісна обробка	Інтерполяція (збільшення) і децимація (зменьшення) частоти дискретизації в багатошвидкісних системах телекомунікації, аудиосистемах

DSP системи реального часу.

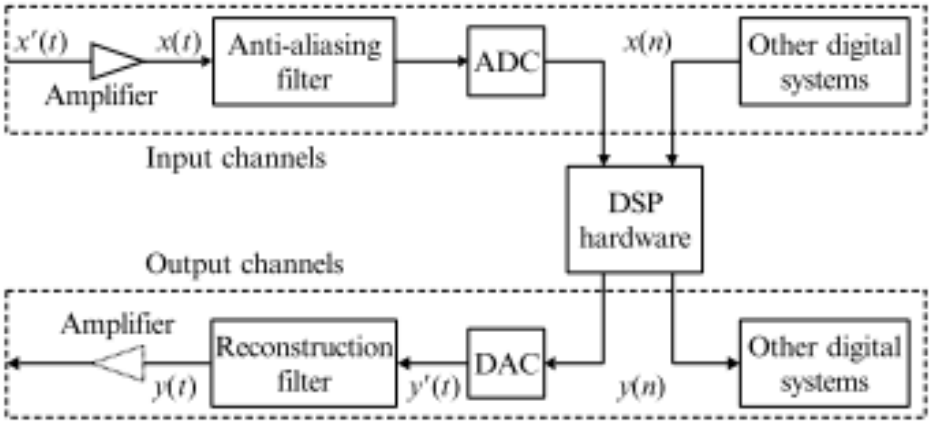


Figure 1.1 Basic functional blocks of real-time DSP system

Цифрова обробка принципово пов'язана з поданням будь-якого сигналу в вигляді послідовності чисел. Це означає, що вхідний аналоговий сигнал повинен бути перетворений у вхідну послідовність чисел (рис. 1.1), яка обчислюється за заданим алгоритмом перетворюється в нову послідовність, однозначно відповідну вихідної. З отриманої нової послідовності формується результуючий аналоговий сигнал.

Основні елементи узагальненої схеми цифрової обробки аналогових сигналів:

аналоговий антіелайсінговий фільтр низьких частот (АФНЧ);

аналого-цифровий перетворювач (АЦП);

пристрій цифрової обробки сигналів (обчислювач);

цифро-аналоговий перетворювач (ЦАП);

аналоговий згладжуючий фільтр низьких частот (ФНЧ).

Антиелайсинговий фільтр формує аналоговий сигнал зі значно зменшеними верхніми частотними складовими (рис. 1.2, г, д) в смузі затримування, починаючи з частоти F=f_v. Це дає підставу вважати сигнал практично обмеженим по частоті і усуває ефект накладання частот.

АЦП виконує дискретизацію та квантування. *Дискретизація* за часом (або дискретизація) являє собою процедуру взяття миттєвих значень сигналу *x( )* через рівні проміжки часу *T*. Миттєві значення x (*nT*) називаються вибірками, або відліками, час T - періодом дискретизації, *n* вказує порядковий номер відліку. *Квантування* відліків за рівнями (або квантування) проводиться з метою формування послідовності чисел: весь діапазон зміни величини відліків розбивається на деяку кількість дискретних рівнів, і кожному відліку за певним правилом присвоюється значення одного з двох найближчих рівнів квантування , між якими виявляється даний відлік. В результаті виходить послідовність чисел *x(nT) = x(n)*, які подаються в двійковому коді.

Максимальна помилка квантування при використанні округлення в якості наближення дорівнює половині кроку квантування. Звідси випливає, що чим більше розрядність АЦП, тим точніше представляється відлік, але тим складніше і дорожче виявляється АЦП, який потрібен для вирішення поставленого завдання. Сучасні АЦП мають розрядність від 8 до 20.

Послідовність $x(nT) = x(n)$ надходить на обчислювач, який по заданому алгоритму кожному відліку $x(n)$ ставить в однозначна відповідність вихідний відлік $y(nT) = y(n)$

Результатом переробки вихідного сигналу є нова цифрова послідовність - цифровий сигнал, який вже не має постійної складової і суттєво відрізняється від $x(n)$. Кількість операцій (множень, додавань, пересилання або т. і.) для отримання одного відліку $y(n)$ може обчислюватися тисячами, тому обчислювач повинен працювати на більш високій тактовій частоті, щоб встигнути провести всі необхідні дії до надходження чергового відліку $x(n)$, тобто якої би складності не був алгоритм, час переробки не повинен перевищувати періоду дискретизації Т.

Але це може бути забезпечено лише в разі, коли тактова частота обчислювача істотно перевищує частоту дискретизації. Саме за цих умов можлива робота обчислювача в реальному часі, тобто в темпі надходження вхідних відліків. Наприклад, при обробці стандартного телефонного сигналу з частотою дискретизації 8 кГц для забезпечення роботи обчислювача в реальному часі тактова частота повинна бути рівною, як найменше, 6 МГц, як в процесорі першого покоління TMS320C10.

Цифровий сигнал також може бути використано іншими пристроями цифрової обробки.

Отримані вихідні відліки подаються на цифро-аналоговий перетворювач (ЦАП), який формує ступінчастий сигнал $y(t)$, котрий потім за допомогою фільтра НЧ перетворюється в аналоговий безперервний сигнал $y(t)$ та підсилюється.

З усього сказаного випливає ряд обмежень, що впливають на характер і вибір елементної бази для реалізації обчислювача:

- розрядність регістрів повинна бути великою і перевищувати розрядність ЦАП щоб уникнути додаткових помилок при округленні результатів обчислень;
- тактова частота, на якій працює обчислювач, повинна в сотні разів перевершувати частоту дискретизації, якщо ставиться вимога реального часу;
- мале енергоспоживання;
- компактність.

Практична реалізація КІХ фільтру у середовищі Arduino

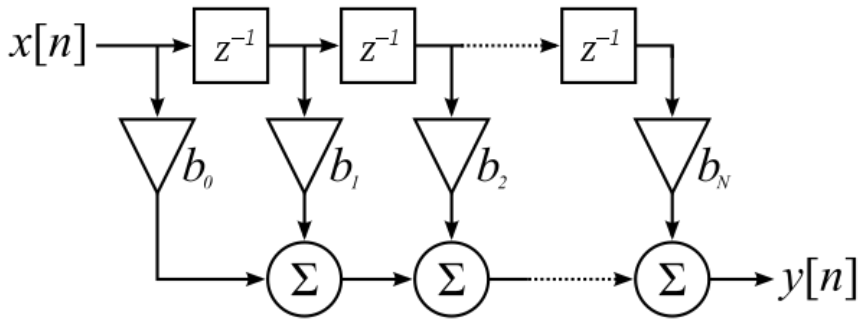


Рис.2 Структурна схема КІХ фільтру.

Формула для розрахунку фільтра:

$$y(n) = \sum_{k=0}^{N-1} (b_k(n) \times x(n-k))$$

Визначимо коефіцієнти для ФНЧ. Вираз для ідеальної імпульсної характеристики ФНЧ – функція ***sinc***

$$h_D(n) = 2f_c \times \frac{\sin(nw_c)}{nw_c} \text{ для } n \neq 0$$
$$h_D(n) = 2f_c \text{ для } n = 0$$

Але нам потрібна «реальна» імпульсна характеристика. Для її розрахунку нам знадобиться вагова функція w (n), їх є декілька різновидів, залежно від вимог до фільтру (Хеммінга, Хеннінга, Блекмена, Кайзера), в нашому випадку використовуємо функцію Блекмена:

$$w(n) = 0{,}42 - 0{,}5 \times cos\left(\frac{2\pi n}{N-1}\right) + 0{,}08 \times \cos\left(\frac{4\pi n}{N-1}\right),$$

де N – кількість коефіцієнтів.

В результаті коефіцієнти фільтру можуть бути обраховані за допомогою наступного виразу.

$$b(n) = h_D(n) \times w(n)$$

Код програми

```
#define N 5
int input = A0;
float coef[FILTERTAPS] = {0.021, 0.096, 0.146, 0.096, 0.021};
float gain = 0.38;
int x[N];
```

```
int out;

void setup() {

     analogReference(INTERNAL1V1);
     Serial.begin(115200);
     //setCoefficients(coef);
     //setGain(gain);
}
void loop() {

    for(int j=0;j<N;j++)    // taking Samples
      x[j]= analogRead(input);

    for(int k=0;k<N;k++)   //Convolution
    {
      out+= coef[k]*x[N-1-k];        //Filter1
    }

    out /= gain;         //Normalisation
    Serial.println(out);
}
```

Контрольні питання

1. Основні напрямки розвитку цифрової обробки сигналів. Прокоментуйте кожен з них.

2. Як побудована системи цифрової обробки сигналів реального часу.

3. Поясніть алгоритм реалізації КІХ фільтру.

4. Поясніть програмний код реалізації КІХ фільтру у середовищі Arduino.

Лекція 15. Подання цілих чисел в мікропроцесорних системах (МПС).

Беззнакові цілі числа подають в МПС натуральним кодом.

Натуральним кодом називають подання числа як *цілого беззнакового* у двійковій системі числення. Діапазон чисел у натуральному коді для *n*-розрядної сітки складає від 0 до $2^n - 1$, тобто для 8-розрядної сітки діапазон чисел у натуральному коді складає від 0 до 255. Наприклад, натуральний код числа 53_{10} (35_{16}) у 8-розрядній сітці наведено на рис 24.6.

Розряди	*D*7	*D*6	*D*5	*D*4	*D*3	*D2*	*D1*	*D*0
Вага розрядів	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
53_{10}	0	0	1	1	0	1	0	1

Рисунок 24.6 - Натуральний код числа 53_{10} у 8-розрядній сітці

Для подання *цілих знакових чисел* використовують ***додатковий код***. Старший розряд сітки є знаковим. Значення цього розряду дорівнює нулю для додатних чисел і одиниці – для від‘ємних. В інших розрядах розміщується модуль числа. *Додатні числа* подаються натуральним кодом. Так, додатне число $+53_{10}$ має вигляд, поданий на рис.24.6. Але оскільки старший розряд є знаковим, діапазон додатних чисел складає від 0 до 2^{n-1}-1. Наприклад, для 8-розрядної сітки діапазон додатних чисел складає від 0 до +127.

Розряди	*D*7	*D*6	*D*5	*D*4	*D*3	*D2*	*D1*	*D*0
Ваги розрядів	-	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	Знак числа	Модуль числа						
	1	1	0	0	1	0	1	1

Рисунок 24.7 - Додатковий код числа -53_{10} у 8-розрядній сітці

Подання *від‘ємного числа* у додатковому коді здійснюється обчисленням числа, яке доповнює додатне число з тим самим модулем до найбільшого беззнакового числа, з подальшим додаванням одиниці до результату. Інакше кажучи, додатковий код отримується шляхом додавання 1 до оберненого (інверсного) коду.

Додатковий код можна одержати за наступним формальним правилом: цифри прямого коду додатного числа необхідно інвертувати послідовно зліва направо до останньої одиниці, не включаючи її. Останню праву одиницю і наступні за нею (праворуч) нулі необхідно залишити без зміни. Діапазон від‘ємних чисел у додатковому коді складає від 0 до -2^{n-1}. На рис. 24.7 показано додатковий код числа -53_{10} у 8-розрядній сітці, для якої діапазон від‘ємних чисел складає від 0 до -128_{10}, при цьому додатковим кодом найменшого числа -128_{10} є число $1000\ 0000_2$.

Особливістю додаткового коду є те, що операцію віднімання війкових чисел можна замінити операцією додавання у додатковому коді.

Перевагами додаткового коду є простота операцій одержання та додавання чисел із різними знаками, а також те, що нуль має єдине подання 0=00000000. Завдяки цим перевагам додатковий код використовується найбільш часто.

Подання дробових чисел в МП Спосіб розміщення розрядів числа в розрядній сітці визначається формою подання двійкових чисел:із фіксованою або з плаваючою комою.

Подання чисел у формі з фіксованою комою. Для розміщення двійкового числа, що містить цілу і дробову частини (без урахування знака) у *n*-розрядній сітці *k* комірок приділяють для розміщення цілої частини та *n*-*k* комірок – для розміщення дробової. При такому поданні двійкових чисел положення коми у розрядній сітці фіксовано.

Подання чисел у формі з плаваючою комою. Форму з плаваючою комою застосовують для розширення діапазону і зменшення відносної похибки подання чисел у МП.

Число *N* зображують у вигляді добутку. Першим множником є правильний дріб *a*, який називається ***мантисою*** числа. Другим множником є основа 2, піднесена до степеня *p*, який називається ***порядком числа***:

$$N = \pm a \cdot 2^{\pm p}.$$

Мантиса і порядок є знаковими числами. Для зазначення знаків у розрядній сітці відводяться 2 додаткових розряди. З такою формою подання існують різні варіанти запису одного і того самого числа. Наприклад, число $11{,}01_2$ можна записати як $0{,}01101_2 \cdot 2^{11_2}$ або як $0{,}1101_2 \cdot 2^{10_2}$. Таким чином, кома у мантисі може зсуватися (плавати), а мантиса може набувати різних значень, менших від одиниці, при відповідних значеннях порядку. Форма подання числа, в якому старший розряд мантиси не дорівнює 0, називається ***нормалізованим***. Всі інші форми подання є ненормалізованими.

У мікропроцесорних системах, в яких реалізовано подання чисел у формі з плаваючою комою, числа зберігають в нормалізованому вигляді, при цьому більша кількість розрядів використовують для зберігання дробової частини, внаслідок чого підвищується точність обчислень. Якщо після виконання арифметичних операцій (наприклад, віднімання) результат виявляється ненормалізованим, то перед занесенням числа в пам'ять виконують його нормалізацію, тобто зсув мантиси ліворуч на відповідну кількість розрядів, і зменшення порядку числа на відповідну кількість одиниць.

При запису двійкового числа у формі з плаваючою комою у (*n*+2)-розрядній сітці *k* комірок приділяють для розміщення мантиси, а *n*-*k* комірок – для розміщення порядку, а 2 розряди – для зазначення знаків (рис.24.8)

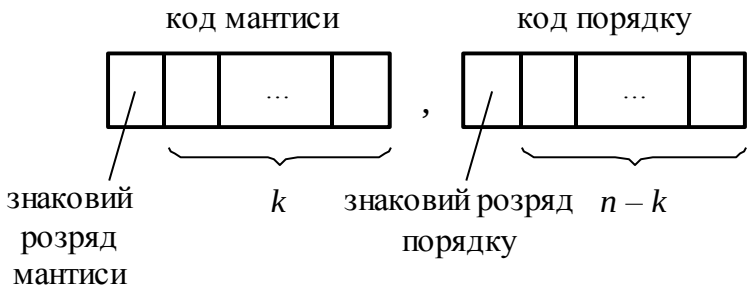


Рисунок 24.8 - Подання чисел у формі з плаваючою комою

Діапазон чисел у формі з плаваючою комою істотно ширший, ніж із фіксованою комою, а відносна похибка значно менше. Форму подання двійкових чисел обирають в залежності від типу задачі, потрібної швидкодії та точності виконання арифметичних операцій та діапазону зміни значень величин, з якими оперує МП.

Цифрова обробка сигналів DSP (Digital Signal Processor)

DSP є спеціалізованими процесорами для додатків, що вимагають інтенсивних обчислень. Якщо ближче розглянути, наприклад, процес операції множення двох чисел із збереженням результату в традиційних мікропроцесорах, то можна побачити як витрачається машинний час: спочатку виконується вибірка команди (адреса команди виставляється на шину адреси), потім першого операнда (адреса операнда виставляється на шину адреси), потім операнд переноситься в акумулятор, далі відбувається вибірка другого операнда і т.д. Прискорення цього процесу в процесорі загального призначення неможлива через наявність єдиної шини адреси і єдиної шини даних, а також єдиного банку даних.

Крім того, якщо розглянути операцію циклічного підсумовування арифметичного ряду, то можна бачити, що тут непродуктивні витрати часу пов'язані із запам'ятовуванням адреси першої команди циклу, з перевіркою умови циклу (лічильника) і поверненням до першої команди.

Також великі непродуктивні витрати існують при операціях переходу до підпрограми і повернення (запис і відновлення значень регістрів із стека) і при багатьох інших операціях. Якщо при цьому врахувати величезну кількість математичних операцій при виконанні цифрової обробки сигналів, то стане ясно, що неминучими є вельми чутливі втрати в точності обчислення при округленнях, які не можуть не позначитися на загальному результаті. Це відбувається внаслідок однакової розрядності всіх регістрів процесорів загального призначення. При цифровій обробці сигналів всі ці витрати неприпустимі.

З метою подолання цих недоліків процесорів загального призначення і були розроблені цифрові процесори обробки сигналів (ЦПОС) або DSP (Digital Signal Processor).

DSP мають *гарвардську архітектуру*. Особливість цієї архітектури полягає, перш за все, в тому, що на відміну від звичних нам двох шин (шини адреси і

шини даних), а також одного банку пам'яті, DSP має, як мінімум, 6-7 різних шин і 2-3 банки пам'яті. Ця особливість має своєю метою максимально прискорити виконання операції множення із збереженням результату, яка, поза сумнівом, є дуже ресурсоємною при цифровій обробці. Архітектура DSP дозволяє за один машинний цикл провести:

- вибірку команди за допомогою шини адреси програм і шини даних програм;
- вибірку двох операндів для операції множення за допомогою двох шин адреси даних;
- занесення операндів в акумулятори за допомогою двох шин даних;
- операцію множення;
- зберегти результат в акумуляторі.

Таким чином, Гарвардська архітектура дозволяє виконати практично будь-яку операцію за один машинний цикл. В якості прикладу ефективності використовування DSP при реалізації алгоритмів цифрової обробки сигналів можна навести такий факт: час виконання комплексного 1024-точкового перетворення Фур'є складає 20 мс для 486DX2 66 Мгц (32-розрядний) і 3,23 мс для 24-розрядного 33 Мгц DSP56001 фірми Motorola або 3,1 мс для 32-розрядного 33 Мгц DSP TMS320C30 з плаваючою арифметикою фірми Texas Instruments. Проте, як вже згадувалося, процесори цифрової обробки сигналу мають своєю особливістю не тільки високу продуктивність, яка вимірюється в швидкості виконання операцій множення/акумуляції (MIPS – мільйони команд в секунду), але і такі характеристики, як послідовність виконання програм, арифметичних операцій і адресації пам'яті, що дозволяє скоротити до мінімуму непродуктивні витрати часу.

Як правило, процесори цифрових сигналів мають *спрощену систему команд*, яка не дозволяє виконати операції, не пов'язані з математичними обчисленнями з такою ж ефективністю, як у процесорів загального призначення. Тому DSP використовують частіше у вигляді сопроцесорів (математичних, графічних, акселераторів і т. д.) при головному процесорі або як самостійний процесор, якщо цього достатньо.

Способи реалізації апаратного забезпечення DSP пристроїв

Table 1.1 Summary of DSP hardware implementations

	ASIC	DBB	μP	DSP chips
Chip count	1	> 1	1	1
Flexibility	none	limited	programmable	programmable
Design time	long	medium	short	short
Power consumption	low	medium–high	medium	low–medium
Processing speed	high	high	low–medium	medium–high
Reliability	high	low–medium	high	high
Development cost	high	medium	low	low
Production cost	low	high	low–medium	low–medium

Контрольні питання

1. Поясніть, як подаються цілі додатьні та від’ємні числа у двійковій системі числення.

2. Поясніть, як подаються дробові числа у двійковій системі числення.

3. Які основні особливості цифрової обробки сигналів.

4. Які є способи реалізації апаратного забезпечення пристроїв цифрової обробки сигналів.

Лекція 16. Організація обчислень в ЦПОС. Особливості архітектури

Процесори ЦПОС можна для зручності розділити на дві категорії: універсальні і спеціалізовані.

Існує два типи спеціалізованих пристроїв.

1. Апаратне забезпечення, розроблене для ефективного виконання спеціальних алгоритмів ЦОС, таких, як цифрова фільтрація, швидке перетворення Фур'є. Пристрої даного типу іноді називають *алгоритмічними процесорами* ЦОС.
2. Апаратне забезпечення, розроблене для спеціального додатку, наприклад, у сфері контролю, телекомунікацій або цифрового аудіо. Пристрої даного типу іноді називають *процесорами ЦОС спеціального призначення* (спеціалізованими) .

В більшості випадків спеціалізовані процесори виконують такі алгоритми ЦОС, як кодування-декодування. Крім того, вони повинні виконувати інші операції, що відображають специфіку додатку.

Всі універсальні і спеціалізовані процесори можна побудувати за допомогою окремих мікросхем або блоків помножувачів, арифметико-логічних пристроїв (АЛП), елементів пам'яті та ін.

Розглянемо архітектурні особливості процесорів ЦОС, які дозволили застосувати цифрову обробку у реальному часі в багатьох областях.

Більшість доступних зараз універсальних процесорів заснована на архітектурі фон Неймана, при якій операції виконуються послідовно. При обробці інструкції в такому процесорі блоки процесора, не задіяні в кожній фазі інструкції, перебувають у стані очікування до передачі їм управління. Підвищення швидкості процесора досягається за рахунок прискорення роботи окремих блоків.

Якщо пристрій має працювати у реальному часі, то архітектуру процесора ЦОС потрібно оптимізувати під виконання функцій ЦОС. Наприклад, на рис. 16.1 показана загальна апаратна архітектура цифрової обробки сигналів у реальному часі. Вона характеризується такими особливостями:

- містить багатошинну структуру з роздільною пам'яттю для даних і інструкцій програми. Звичайно пам'ять для зберігання даних містить вхідні дані, проміжні значення і вихідні вибірки, а також фіксовані коефіцієнти, наприклад, для цифрової фільтрації або ШПФ. Команди програми зберігаються у спеціально відведених елементах пам'яті;
- порт вводу-виводу дозволяє обмінюватися даними із зовнішніми пристроями (АЦП, ЦАП) або передавати цифрові дані іншим процесорам. Прямий доступ до пам'яті (Direct Memory Access — DMA), якщо він є, дозволяє швидко обмінюватися блоками даних з пам'яттю (ОЗП) для зберігання даних, причому звичайно це відбувається під зовнішнім управлінням;

• містить арифметичні пристрої для логічних і арифметичних операцій, до числа яких виходять АЛП, апаратні помножувачі і схеми зсуву (або помножувачі-накопичувачі).

Для цифрової обробки сигналів використовуються так звані сигнальні мікропроцесори. До їх особливостей можна віднести малорозрядну (40 розрядів і менш) обробку чисел з плаваючою точкою, переважне використання чисел з фіксованою точкою розрядності 32 і менш, а також орієнтацію на нескладну обробку великих масивів даних.

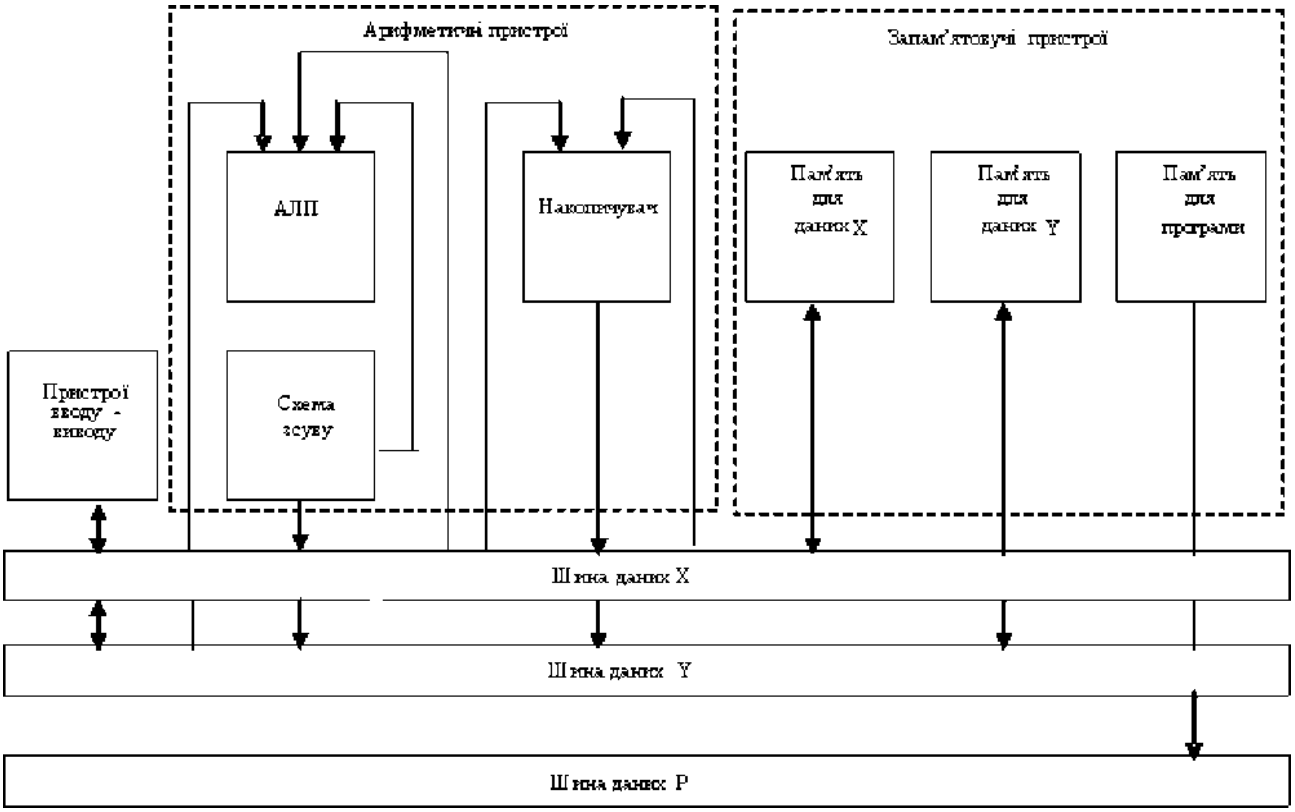


Рис. 16.1. Стандартна універсальна апаратна архітектура обробки сигналів

Відміною особливістю задач цифрової обробки сигналів є потоковий характер обробки великих обсягів даних в реальному режимі часу, який вимагає від технічних засобів високої продуктивності і забезпечення можливості інтенсивного обміну із зовнішніми пристроями. Відповідність до даних вимог досягається завдяки специфічній архітектурі сигнальних процесорів і проблемно-орієнтованій системі команд.

Сигнальні процесори мають високий ступінь спеціалізації. У них широко використовуються методи скорочення тривалості командного циклу, характерні і для універсальних RISC-процесорів:

- конвеєризація на рівні окремих мікроінструкцій і інструкцій,
- розміщення операндів більшості команд в регістрах,
- використання тіньових регістрів для збереження стану обчислень при перемиканні контексту,
- розділення шин команд і даних (гарвардська архітектура).

У той же час для сигнальних процесорів характерною є наявність апаратного помножувача, що дозволяє виконувати множення двох чисел за один командний такт. В універсальних процесорах множення звичайно реалізується за декілька тактів, як послідовність операцій зсуву і складання. Іншою особливістю сигнальних процесорів є включення до системи команд таких операцій, як множення з накопиченням MAC (C:=AxB+C з вказаним в команді числом виконань в циклі і з правилом зміни індексів елементів масивів A і B), інверсія біт адреси, різноманітні бітові операції. У сигнальних процесорах реалізується апаратна підтримка програмних циклів, кільцевих буферів. Один або декілька операндів витягуються з пам'яті в циклі виконання команди.

Реалізація однотактного множення і команд, що використовують як операнди вміст елементів пам'яті, обумовлює порівняно низькі тактові частоти роботи цих процесорів. Спеціалізація не дозволяє підіймати продуктивність за рахунок швидкого виконання коротких команд типу «регістр-регістр», як це робиться в універсальних процесорах. Цих команд просто немає в програмах обробки сигналів.

MAC-операція. Схеми реалізації в ЦПОС. Точність обчислень

Основними чисельними операціями в цифровій обробці сигналів є множення і складання. Множення в програмній формі сумно відоме своєю трудомісткістю, а якщо використовується арифметика з плаваючою точкою, складання виявляється навіть ще більш трудомісткою операцією. Щоб максимально прискорити цифрову обробку сигналів у реальному часі, рекомендується використовувати спеціалізовані апаратні помножувачі-накопичувачі для арифметики з плаваючою або фіксованою точкою. Таке апаратне забезпечення тепер стандартно використовується у всіх цифрових процесорах сигналів. У процесорі з фіксованою точкою апаратний помножувач за один такт (звичайно 25 нс) приймає два 16-бітові дробові числа, подані у формі доповнення до двох, і обчислює їх 32-бітовий добуток. Середній час виконання команди множення-накопичення можна значно зменшити, якщо використовувати спеціальні команди повторення.

Типова конфігурація апаратного помножувача-накопичувача ЦОС зображена на рис. 3.7. У такій конфігурації помножувач має пару вхідних регістрів (Рг X та Рг Y), які містять входи помножувача, і 32-бітовий регістр добутку (Рг P), який містить результат множення. Вихід регістра P (product - добуток) з'єднується з накопичувачем подвійної точності, в якому накопичуються добутки.

Подібна конфігурація застосовується в апаратних помножувачах-накопичувачах з плаваючою точкою, за винятком того, що входи і добутки - це нормовані числа, що подані у формі з плаваючою точкою. Помножувачі-накопичувачі з плаваючою точкою дозволяють швидко обчислювати результати алгоритмів ЦОС з мінімальним помилками.

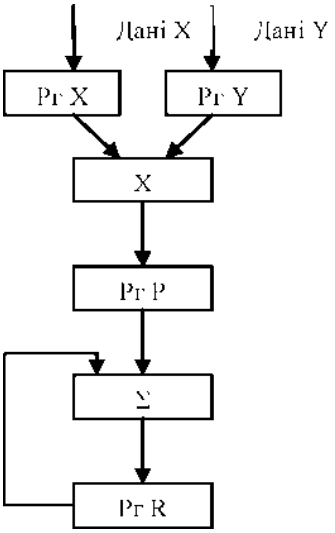


Рис. 3.7. Типова конфігурація апаратного помножувача-накопичувача ЦОС

Операції з плаваючою точкою дозволяють використовувати великий динамічний діапазон і знижують арифметичні помилки, хоча в багатьох додатках достатньо динамічного діапазону, який пропонує подання з фіксованою точкою.

Двома найпоширенішими типами арифметики, що використовуються в сучасних ЦПОС, є арифметики з фіксованою і плаваючою точками.

Арифметика *з плаваючою точкою* - це природний вибір в додатках з широкими і змінними вимогами до динамічного діапазону (динамічний діапазон можна визначити як різницю між найбільшим та найменшим рівнем сигналу, який можна представити, або як різницю між найбільшим сигналом і мінімальним рівнем шуму, що вимірюються в децибелах).

Процесори з *фіксованою точкою* переважні з погляду низької вартості, вони підходять для масового виробництва (наприклад, стільникові телефони і комп'ютерні дисководи). При використанні арифметики з фіксованою точкою виникають питання, пов'язані з обмеженнями динамічного діапазону. Взагалі процесори з плаваючою точкою дорожчі за процесори з фіксованою точкою, хоча останніми роками різниця в ціні істотно зменшилася. Більшість існуючих ЦПОС з плаваючою точкою також підтримує арифметику з фіксованою точкою.

Використання даних у форматі з плаваючою точкою в сигнальній обробці обумовлене декількома причинами. Для багатьох задач, пов'язаних з виконанням інтегральних і диференціальних перетворень, особливе значення має точність обчислень, забезпечити яку дозволяє експоненціальний формат подання даних. Алгоритми компресії, декомпресії, адаптивної фільтрації в ЦОС пов'язані з визначенням логарифмічних залежностей і досить чутливі до точності подання даних в широкому динамічному діапазоні.

Робота з даними у форматі з плаваючою точкою істотно спрощує і прискорює обробку, підвищує надійність програми, оскільки не вимагає

виконання операцій округлення і нормалізації даних, відстежування ситуацій втрати значущості і переповнювання.

Фізична організація пам'яті в ЦПОС. Гарвардська архітектура та багатопортова пам'ять

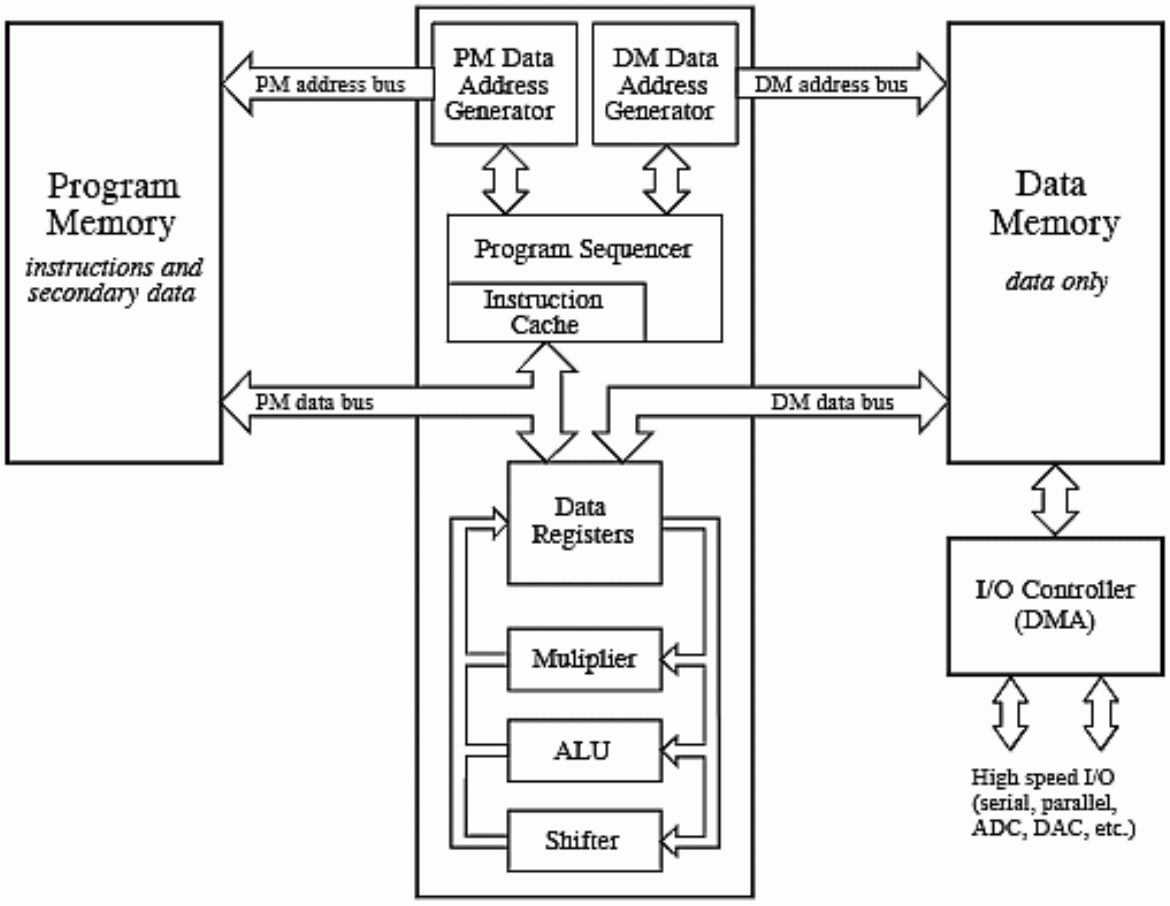


FIGURE 28-5
Typical DSP architecture. Digital Signal Processors are designed to implement tasks in parallel. This simplified diagram is of the Analog Devices SHARC DSP. Compare this architecture with the tasks needed to implement an FIR filter, as listed in Table 28-1. All of the steps within the loop can be executed in a single clock cycle.

Рис. 3.8. ЦПОС архітектура

Фізично пристрої пам'яті в ЦПОС можна розподілити на внутрішні та зовнішні. До внутрішніх належать:

- оперативний запам'ятовуючий пристрій (ОЗП), присутній у всіх типах ЦПОС;
- постійний запам'ятовуючий пристрій (ПЗП), присутній у деяких типах ЦПОС;
- напівпостійний запам'ятовуючий пристрій (НПЗП), присутній у деяких типах ЦПОС.

Ємність ОЗП, ПЗП та НПЗП залежить від типу.

Зовнішню пам'ять можна розподілити:

- на пам'ять для збереження даних;
- пам'ять для збереження програм.

Принципова особливість гарвардської (двошинної) архітектури полягає у тому, що пам'ять для зберігання даних і пам'ять для зберігання програми розташовуються в різних місцях, припускаючи повне поєднання в часі операцій виклику команди з пам'яті і її виконання [58]. Стандартна гарвардська архітектура наведена на рис. 3.8. Звичайні мікропроцесори, такі, як Intel 6502, характеризуються одношинною структурою, через яку передаються і дані, і команди. У гарвардській архітектурі, де команди програми і дані зберігаються в різних областях пам'яті, виклик наступної команди може співпадати з виконанням поточної команди. Як правило пам'ять програми містить програмний код, тоді як пам'ять для зберігання даних включає змінні, наприклад, вибірки вхідних даних.

Стандартна гарвардська архітектура використовується лише в декількох процесорах ЦОС (наприклад, Motorola DSP56000). Відповідна схема реалізації доступу до пам'яті має один очевидний недолік - високу вартість. При поділі каналів передачі команд і даних на кристалі процесора останній повинен мати майже вдвічі більше інтерфейсних виводів, так як шина адреси і шина даних складають основну частину виводів мікропроцесора.

Способом вирішення цієї проблеми стала ідея використовувати загальні шину даних і шину адреси для всіх зовнішніх даних, а всередині процесора використовувати шину даних, шину команд і дві шини адреси. Таку концепцію стали називати модифікованої гарвардської архітектурою. У модифікованій архітектурі також виділяються роздільні області пам'яті для зберігання програми і даних, але на відміну від строгої гарвардської архітектури тут дозволений зв'язок між двома областями пам'яті. В більшості пристроїв застосовується *модифікована гарвардська архітектура* (наприклад, сімейство процесорів TMS320)

Для взаємодії із зовнішніми пристроями (АЦП, ЦАП та ін.) або з іншими процесорами в ЦПОС передбачена система портів вводу-виводу. Спосіб передачі даних (послідовний чи паралельний) та розрядність номенклатура портів вводу-виводу залежить від конкретного типу ЦПОС.

Логічна організація пам’яті та режими адресації

Комбінація гарвардської архітектури з швидкодіючою та багатопортовою пам’яттю дозволяє реалізувати вимоги до ефективної організації взаємодії з пам’яттю, якої потребують алгоритми ЦОС.

Деякі ЦПОС мають спеціальний кеш команд, який є пам’яттю невеликого розміру, що входить до складу ядра процесора та призначається для

збереження команд. Попереднє завантаження команд до кеша дозволяє звільнити шини процесора для доступу до даних. Найпростіші процесори мають кеш на одну команду. В більш складних ЦПОС ємність кеша складає 1К слів.

При виконанні будь-якої команди процесор звертається до пам'яті програми або до пам'яті даних. Спосіб визначення адреси операнда в команді або адреси передачі управління називається режимом адресації. У ЦПОС використовують режими таких видів адресації: прямої, безпосередньої та різних видів непрямої адресації.

Пряма адресація. Абсолютна адреса операнда міститься в кодовому слові команди. Наприклад, команда ЦПОС ADSP-21xx

AX0=DM(0800)

завантажує значення, розташоване за абсолютною адресою 0800 пам'яті даних до регістру AX0.

Безпосередня адресація. Значення операнда міститься в команді. Наприклад, команда ЦПОС ADSP-21xx

AX0=1357

Значення константи 1357 завантажується до регістру AX0. Якщо значення константи перевищує довжину одного кодового слова команди, воно розміщується в наступному кодовому слові. Це призводить до читання з пам'яті двох слів, що підвищує час виконання програми.

Непряма адресація. Використовує регістри-покажчики, вміст яких є реальною адресою розміщення даних у пам'яті. У цьому випадку в команді міститься тільки посилання на регістр-покажчик. Кількість регістрів для непрямої адресації в ЦПОС невелика, тому довжина команди непрямої адресації дорівнює одному слову. У ЦПОС використовуються такі види непрямої адресації: непряма регістрова адресація, непряма регістрова адресація з інкрементуванням, непряма регістрова адресація з модульною арифметикою, біт-інверсна адресація. Проілюструємо ці види адресації за допомогою операторів мови С++.

Непряма регістрова адресація. Відповідає наступній формі оператора присвоєння:

A=A+*R;

Відповідно до цього оператора значення, що зберігається за адресою пам’яті, яка міститься в регістрі R, додається до значення акумулятора A. Регістр R розглядається як покажчик.

Непряма регістрова адресація з інкрементуванням та декрементуванням. Відповідає наступним операторам:

A=A+(*R)++;

A=A+(*R)--;

Значення, що зберігається за адресою пам’яті, яка міститься в регістрі R, додається до значення акумулятора A. Після отримання значення з пам’яті вміст регістру-покажчика збільшується (++) або зменшується (--) на одиницю. Деякі ЦПОС дозволяють збільшувати або зменшувати вміст регістра-покажчика не на одиницю, а на іншу величину, значення якої зберігається в додатковому регістрі.

Непряма регістрова адресація з модульною арифметикою. Використовується при організації в пам’ яті кільцевого буфера. Буфер - послідовність однотипних елементів даних, що розташовані у суміжних комірках пам’яті. Якщо елементи даних обробляються в порядку їх запису у пам’ ять, то буфер відповідає структурі даних, яка називається чергою. Для доступу до значень, що зберігаються в буфері, використовуються покажчики. Кожен раз після звертання до буфера значення покажчика змінюється з визначеним кроком. При досягненні кінця буфера значення покажчика змінюється на початкове. Цей механізм можна описати за допомогою умовного оператора

if(pointer+step<BkEnd);

pointer=pointer+step;

else pointer=BkBegin;

Змінна BkBegin визначає початкову адресу, а BkEnd - кінцеву адресу області пам’яті, в якій розміщено буфер. Змінна pointer - покажчик, а змінна step відповідає кроку, з яким змінюється значення покажчика. Ця схема адресації використовується, наприклад, в ЦПОС TMS320C5x.

Можливе використання іншого механізму адресації кільцевого буфера, який застосовується в ЦПОС ADSP21xx, ADSP21xxx, TMS320C3x, TMS320C4x. У цих ЦПОС не задається кінцева адреса області пам’яті, відведеної під буфер, а в спеціальному регістрі фіксується довжина буфера. Значення, що відповідає довжині буфера, називається модулем. При збільшенні

початкової адреси на величину модуля здійснюється перехід до початкової адреси, що і складає суть непрямої адресації з модульною арифметикою.

Біт-інверсна адресація. Дозволяє змінювати впорядкованість вхідних або вихідних даних, що потрібно, наприклад, в алгоритмах ШПФ. Біт- інверсний порядок задається шляхом «дзеркального» відображення двійкових розрядів вхідної чи вихідної послідовності.

Приклад реалізації наведено в табл.. 3.1

Таблиця 3.1 - Біт-інверсний порядок

N	Двійковий номер	Біт-інверсія	Біт-інверсний
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Архітектура й призначення спеціалізованих модулів ЦПОС: таймера, лічильника числа повторень, генераторів адреси

Призначення та роботу спеціалізованих модулів розглянемо на прикладі ЦПОС сімейства TMS320C2x [60].

У складі ЦПОС є 16-розрядний таймер (TIM) та регістр періоду таймера (PRD). Таймер управляється відповідним сигналом (CLKOUT1). При скиданні таймера в нього та регістр періоду передається максимальне число FFFFh. Після дії сигналу скидання вміст таймера зменшується на одиницю при кожному надходженні сигналу CLKOUT1. Початкове значення таймера визначається вмістом регістру періоду. Як тільки значення в таймері стане дорівнювати нулю, формується переривання від таймера (TINT) і він перезавантажує те значення, що знаходиться в регістрі періоду. Це відбувається під час першого машинного циклу після формування переривання TINT. Таким чином, переривання будуть відбуватися через проміжки часу, що дорівнюють $[(PRD)+1]*T_{CLKOUT1}$, де $T_{clkout1}$ - період сигналу CLKOUT1.

Зазвичай переривання TINT використовується для зчитування вибірок вхідних даних, що обробляються процесором. Таймер та регістр періоду можуть бути проініціалізовані в будь-якому машинному циклі. Нульове значення регістру періоду не допускається.

Якщо переривання TINT не використовуються, то необхідно застосувати маску або заборонити переривання командою DINT. У цьому випадку регістр періоду може використовуватися як звичайна комірка пам’яті. Якщо знову виникне потреба у використанні переривання TINT, то регістр PRD та таймер необхідно попередньо ініціалізувати, а потім дозволити переривання TINT.

Лічильник числа повторень RPTC містить 8-розрядне число N, яке визначає повтори деякої команди. Кількість повторів дорівнює N+1. За допомогою команди RPT або RPTK лічильник числа повторень може бути завантажений значенням з діапазону від 0 до 255. Це дозволяє повторювати команду, що знаходиться за RPT або RPTK до 256 разів.

Лічильник числа повторень може використовуватися з командами: множення з накопиченням, пересилання блоків, вводу-виводу, читання- запису таблиць. Ті команди, що потребують для виконання декількох циклів, при запису їх після команди RPT або RPTK виконуються за один цикл.

Для формування адрес при звертанні до пам’ яті в ЦПОС застосовуються генератори адреси, які можуть формувати одну чи декілька адрес даних за один цикл команди, не використовуючи для цього основного арифметичного пристрою, що займається обробкою даних. Це дозволяє обчислювати необхідні адреси даних паралельно з виконанням арифметичних операцій, що підвищує продуктивність ЦПОС

Контрольні питання

1 Поясніть, які типи спеціалізованих процесорів використовуються для цифрової обробки сигналів.

2. Яку архітектуру мають ЦПОС і яким чином у них організована фізична пам’ять.

3. Яким чином у ЦПОС організована логічна пам’ять та режими адресації.

4. Яке призначення спеціалізованих модулів ЦПОС.

Лекція 17. Внутрішньокристальні емулятори. JTAG-емулятор

Останнім часом компанії - виробники мікропроцесорів і мікроконтролерів інтегрують у свої нові розробки модулі з функціями вбудованої налагодження - відладчики на кристалі. Для підтримки функцій вбудованої налагодження необхідний спеціальний канал зв'язку з комп'ютером. Найчастіше в якості такого каналу використовується тестовий послідовний інтерфейс типу JTAG, тому апаратний засіб налагодження, побудований на базі вбудованого в кристал налагоджувального модуля, часто називають JTAG-емулятором. Абревіатура JTAG виникла по найменуванню розробника - об'єднаної групи по тестам JoINT Test Action Group.

Інтерфейсом JTAG, реалізованим відповідно до стандарту IEEE 1149.1-2001, управляє один пристрій-контроллер (найчастіше це персональний комп'ютер, оснащений відповідним програмним забезпеченням і інтерфейсним адаптером), до якого може бути підключено кілька тестованих пристроїв.

До складу інтерфейсу JTAG входять 5 односпрямованих послідовних ліній зв'язку (один із сигналів необов'язковий). Ці сигнали утворюють тестовий порт TAP (Test Access Port), через який тестоване пристрій підключається до тестує обладнання (контролеру).

У завдання тестуючого устаткування входить формування тестових сигналів по програмі тестування, визначеною розробником тестованого пристрою, і порівняння отриманих результатів з еталонами.

Один і той же контролер і порт можуть використовуватися для тестування будь-якого числа пристроїв, що підтримують JTAG. Для цього пристрою своїми портами TAP з'єднуються в ланцюжок. Стандартизований логічний формат дозволяє контролеру незалежно спілкуватися з кожним з пристроїв і виконувати наступні дії:

- Покроковий (на рівні машинних команд) режим виконання програми.
- Заморожування периферії при зупинці: в момент переходу в режим зупинки
- виконання програми користувача JTAG-емулятор блокує джерело тактової частоти, керуючий роботою центрального процесора і периферійних пристроїв. Це забезпечує точне відстеження модельного часу. JTAG-емулятор забезпечує точну відповідність часу виконання програми користувача часу роботи периферійних пристроїв.

- Доступ до ресурсів мікроконтролера при зупинці: JTAG-емулятор надає доступ до всіх ресурсів мікроконтролера в режимі зупинки виконання програми користувача.

Оскільки інформація вводиться в налагоджуваний пристрій і виводиться з нього через послідовний порт, метод JTAG не може служити заміною повнофункціональному емулятор, так як не в змозі відобразити поточний стан внутрішніх шин.

Однак він має ряд очевидних переваг:

- Низька вартість засоби налагодження: в найпростішому випадку JTAG емулятор може складатися з кабелю, що з'єднує порт комп'ютера з JTAG-портом на наладжуваної платі.
- Всі режими налагодження мікроконтролера підтримуються програмним забезпеченням. Максимально точна відповідність умов налагодження робочим умовам серійного виробу: налагодження МПС може здійснюватися на серійній платі, доповненої тільки 5-контактним роз'ємом порту TAP на серійному МП. Як наслідок, електричні і тимчасові характеристики системи в процесі налагодження абсолютно ідентичні характеристикам робочого режиму.
- Можливість одночасного тестування декількох пристроїв, об'єднаних JTAG-колами, причому стандарт не вводить ніяких обмежень на кількість пристроїв у колі.

До недоліків JTAG-емулятора відносяться наступні:

- Необхідна умова використання JTAG-емулятора - наявність вбудованої в мікроконтроллер Flash-пам'яті програм, оскільки для завантаження програми користувача при налагодженні JTAG-емулятор задіює власну пам'ять мікроконтролера. У мікроконтролерах, у яких застосовується інший тип пам'яті програм (наприклад, тільки ППЗП), реалізація JTAG-емулятора на кристалі неможлива.
- JTAG-емулятор має обмежене (зазвичай не більше 8) кількість точок зупину.
- Труднощі в реалізації покрокового режиму на рівні операторів мови високого рівня, так як для відпрацьовування цього режиму в вихідний текст відладжуваної програми на початку кожного оператора ЯВУ повинна вставлятися команда виклику відладжуваного монітора, а кількість контрольних точок обмежено.
- Програма користувача, яка завантажується для налагодження, має більший розмір і більшу часів виконання, ніж робоча програма, у зв'язку з необхідністю вставки в неї команд виклику відладжуваного монітора. Це порушує, хоча й несуттєво, масштаб реального часу

виконання програми користувача в режимі налагодження з JTAG-емулятором.

- JTAG-емулятор не підтримує точки останову по складним (комплексним) умовам, які реалізовані, наприклад, під емуляторах, які знаходяться в схемах. Широко використовується для відладки реальних пристроїв. Тестована програма може бути тією остаточною версією (після видалення команд виклику монітора налагодження), яка буде поставлятися.

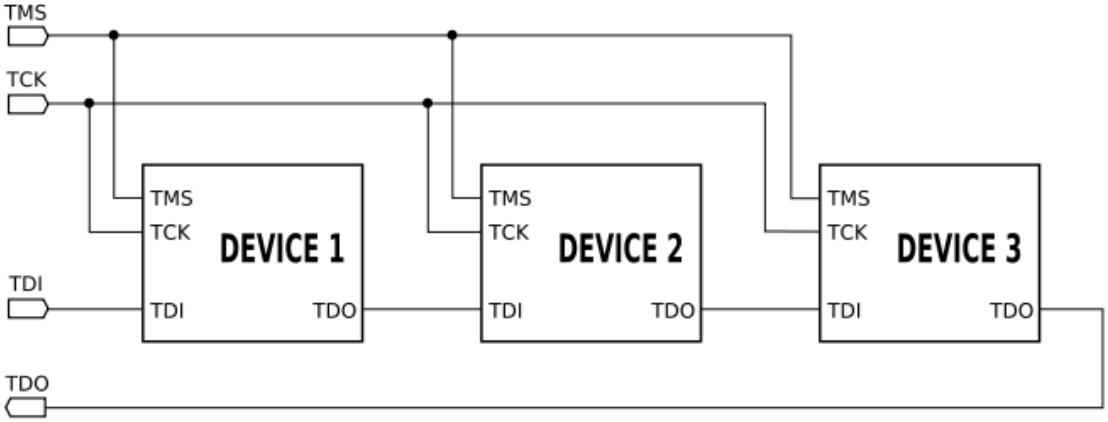
Інтерфейс JTAG застосовується не тільки для тестування, але і для програмування різних пристроїв, у тому числі і незалежної пам'яті мікроконтроллерів. Контакти для сигналів JTAG є на шині PCI, проте в їх використанні одноманітності не спостерігається (або залишаються непідключеними, або з'єднуються для організації ланцюжка).

Відзначимо, що інтерфейсом JTAG в даний час оснащуються не тільки однокристальні мікроконтролери та сигнальні процесори, але і універсальні мікропроцесори аж до моделей з найвищою на сьогоднішній день продуктивністю, наприклад, Intel Core i7.Edition. Тут він дозволяє не тільки тестувати сам процесор (це не представляє особливого прикладного інтересу), але і організувати зондовий режим відладки (probe mode). Зондовий режим є потужним засобом налагодження системного програмного забезпечення; звичайний процесор, пов'язаний з тестовим контролером інтерфейсом JTAG, перетворюється у внутрісхемний емулятор.

Порт тестування (англ. *TAP — Test Access Port*) являє собою чотири або п'ять виділених виводів мікросхеми: TCK, TMS, TDI, TDO і (опціонально) TRST.

- TDI (test data input — «вхід тестових даних») — вхід послідовних даних периферійного сканування. Команди і дані вводяться в мікросхему з цього виводу по передньому фронту сигналу TCK;
- TDO (test data output — «вихід тестових даних») — вихід послідовних даних. Команди і дані виводяться з мікросхеми з цього виводу по задньому фронту сигналу TCK;
- TCK (test clock — «тестове тактування») — тактує роботу вбудованого автомата управління периферійним скануванням. Максимальна частота сканування периферійних осередків залежить від використовуваної апаратної частини і на даний момент обмежена 25 … 40 МГц ;
- TMS (test mode select — «вибір режиму тестування») — забезпечує перехід схеми в / з режиму тестування і перемикання між різними режимами тестування.

Стандарт передбачає можливість підключення великої кількості пристроїв (мікросхем) через один фізичний порт (з'єднувач). Це можливо завдяки послідовному ввімкненню пристроїв та роботі з ними по заданому номеру у ланцюжку.



У деяких випадках до перерахованих сигналів додається сигнал TRST для ініціалізації порту тестування, що необов'язково, оскільки ініціалізація можлива шляхом подачі певної послідовності сигналів на вхід TMS.

Робота засобів забезпечення інтерфейсу JTAG підкоряється сигналам автомата управління, вбудованого в мікросхему. Стан автомата визначаються сигналами TDI і TMS порту тестування. Певне поєднання сигналів TMS і TCK забезпечує введення команди для автомата і її виконання.

Якщо на платі встановлено кілька пристроїв, що підтримують JTAG, вони можуть бути об'єднані в загальну послідовність. Унікальною особливістю JTAG є можливість програмування не тільки самого мікроконтролера (або ПЛІС), але і підключеної до його виводів мікросхеми флеш-пам'яті. Причому існує два способи програмування флеш-пам'яті з використанням JTAG: через завантажувач з подальшим обміном даними через пам'ять процесора, або через пряме управління виводами мікросхеми.

Засоби розробки та налагодження програмного забезпечення.

Для розробки програмного забезпечення в даний час використовуються інтегровані середовища розробки, що містять у своєму складі текстові редактори, компілятори, редактори зв'язків, завантажувачі та симулятори.

Текстові редактори служать для створення тексту програми. Як правило, вони мають більш обмежені можливості в порівнянні з універсальними програмами цього типу і орієнтовані на особливості написання програм на використовуваних мовах програмування.

Застарілою мовою програмування для мікроконтроллерних систем, що працюють в реальному масштабі часу (а саме до цього класу належить

більшість систем керування, збору і обробки інформації на базі однокристальних мікроконтролерів і ЦСП), є Асемблер. В даний час в розпорядження розробників практично повсюдно надається також компілятор з мови С, а іноді С++ і навіть Паскаля. Як правило, в цьому випадку використовуються спеціальні оптимізують компілятори, але навіть вони не завжди дозволяють написати прийнятну за часом виконання і обсягом пам'яті програму, що іноді викликає необхідність їх доопрацювання на Асемблері.

Редактор зв'язків збирає єдиний виконуваний модуль з декількох об'єктних програмних модулів. Якщо програми попередніх типів використовуються при будь-якому процесі програмування, то симулятори є специфічним програмним засобом, використовуваним в процесі проектування МПС. Симулятори надають користувачеві можливість виконати тестування і налагодження розробленого програмного забезпечення на програмно-логічної моделі мікропроцесора. Симулятори дозволяють запустити програму і повністю простежити її виконання. Завантаживши програму в симулятор, користувач має можливість запускати її в покроковому чи безупинному режимах, задавати умовні або безумовні точки зупинки, контролювати і вільно модифікувати вміст комірок пам'яті і регістрів модельованого мікропроцесора.

Симулятор охоплює відразу кілька процесорів одного сімейства. Вибір конкретного типу МП серед моделей сімейства забезпечується відповідними опціями меню. При цьому моделюється робота ЦП, всіх портів введення/ виводу, переривань і іншої периферії.

Спочатку налагодження програм з використанням симуляторов велася на рівні машинних команд в символьних позначеннях регістрів. До складу сучасних симуляторів входять також отладчики на мовах високого рівня, оскільки в комплект розробника, як правило, входить і відповідний компілятор. Основна перевага симуляторів полягає в тому, що, оскільки вони не вимагають наявності реальних апаратних засобів, розробка програмного забезпечення може йти паралельно з їх розробкою.

Головним недоліком цього підходу є те, що, оскільки моделювання здійснюється програмним способом, відлагоджувати програми виконується не в реальному масштабі часу. При цьому всі сигнали вводу/виводу повинні генеруватися спеціальними підпрограмами, розробленими для імітації периферійних пристроїв. Однак існує думка, що добре написаний симулятор дає досить точне уявлення про роботу програми цільового МП, включаючи її часові характеристики.

Спочатку симулятори створювали самі розробники МП БІС і продавали їх за дуже низькою ціною або навіть поставляли безкоштовно, для того щоб потенційні користувачі могли заздалегідь познайомитися з особливостями нових схем і почати розробку ПЗ для них до появи на ринку достатньої кількості нових БІС. Нині симулятори поставляють безліч виробників емуляторів і компіляторів, в той час як традиційні постачальники - виробники інтегральних схем - воліють залишати цей ринок.

Контрольні питання

1. Призначення та принцип роботи JTAG-емулятора.

2. Електрична реалізація JTAG-емулятора.

3. Переваги використання JTAG-емулятора.

4. Яки є засоби розробки та налагодження програмного забезпечення.

Лекція 18. ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ (у вигляді тезисів)

Етапи програмування

- *Створення алгоритму роботи*
- *Створення електричної схеми*
- *Створення коду програми*
  - Мови програмування
  - Середовище розробки
  - Бібліотеки
  - Операційні системи для мікроконтролерів
- *Компіляція*
- *Симуляція*
  - Симуляція в схемі
  - Симуляція
- *«Прошивка» мікроконтролера*
  - Інтерфейси програмування
- *Налагодження*
  - Налагодження в схемі
  - Налагодження в симуляторі
- *Оптимізація*
  - Вимірювання продуктивності
  - Профілювання
  - Рефакторинг коду

Створення алгоритму

- *В значній мірі ручний процес. Є засоби автоматизованої розробки UML (Unified Modeling Language)*
  - Ефективні лише для дуже складних проектів
  - Для простих проектів не ефективні
- *Створення блок-схеми (діаграми)*
  - Дуже допомагає на перших етапах
- *Розрахунок параметрів компонентів блок-схеми*
  - Тривалості, затримки, частоти
  - …
- *Створення алгоритмів роботи компонентів блок-схеми*

Створення блок-схеми

- *Декомпозиція*
  - Розбити задачу на простіші під-задачі
  - Розбити систему на простіші під-системи
  - Уникати дублювання функцій
- *Зв'язок*
  - Як взаємодіють між собою частини підсистеми

- Уникати дублювання функцій
- *Синхронізація*
  - Гарантування того, що певні підсистеми знаходяться в певному визначеному стані

Приклад блок-схеми

- *Система керування світлодіодом*
  - Натиснення кнопки – перемкнути світлодіод
- *Декомпозиція*
  - Блок налаштування периферії для кнопки
  - Блок налаштування периферії для світлодіода
  - Система реагування на події натиснення кнопки
  - Система ввімкнення світлодіода
  - Система прийняття рішення про ввімкнення чи вимкнення
- *Зв'язок – послідовність дій*
- *Синхронізація*
  - Не чіпати світлодіод поки кнопка не змінюється

Створення електричної схеми

- *Декомпозиція, зв'язок синхронізація*
  - Підбір елементів
  - З'єднання
- *Підбір режимів роботи пристроїв*
  - Електричний (напруги та струми)
  - Часовий (тривалості імпульсів, частоти)
  - Обов'язково передбачити обмеження струму резисторами
- *Підбір параметрів елементів*
  - Щоб задовольнити всім технічним вимогам

Створення коду програми

- *Мови програмування*
  - Багато варіантів
  - …
  - Асемблер
  - C/C++
- *Середовища розробки*
  - Для різних мікроконтролерів свої
- *Бібліотеки*
  - Готовий код для різних задач
- *Операційні системи*
  - Програмні засоби віртуалізації ресурсів мікроконтролера

Мова асемблер(а)

- *Одна команда мови відповідає одній команді процесора*
  - Пересилання даних
    - Регістр-регістр, пам’ять-регістр,регістр-пам’ять MOV
    - Запис читання стеку PUSH, POP
  - Логічні операції
    - Set bit, clear bit BIS,BIC
    - AND,OR,XOR,NAND,NOR,XNOR
  - Арифметичні операції
    - ADD,SUB
  - Перевірка бітів (Test)
    - BIT
    - Встановлення прапорця переносу C у випадку успіху
  - Операції переходу по мітці *JMP* мітка
    - Коли прапорці встановлено, чи не встановлено JC,JNC,JZ,JNZ
  - Виклик функцій *CALL, RET*
  - …
- *У кожного типу процесора свій асамблер*
- *Різним регістрам периферії відповідає мнемонічне позначення*
  - P1IN,P1OUT,P1REN ..

Приклад програми на мові асамблер

```
;led
        BIS.B   #11111111b,&P1DIR
        BIC.B   #11111111b,&P1OUT
;button
        BIC.B   #00001000b,&P1DIR
        BIS.B   #00001000b,&P1REN
        BIS.B   #00001000b,&P1OUT
cont1:
       ; адреса ділянки програми
        bit.b #00001000b,&P1IN
        jc cont1
```

```
        XOR.b #00000001b,&P1OUT
cont2:
       ; адреса ділянки програми
        bit.b #00001000b,&P1IN
        jnc cont2
        JMP cont1
```

Мова C/C++

- *Підтримуються високорівневі конструкції*
  - Цикли while, for
  - Умовні переходи if, then, else
  - Вирази
  - Виклики функцій
- *Регістри керування периферією відображаються на пам'ять*
  - Мають визначені адреси P1OUT, P1IN, P1REN
  - Виступають в ролі змінних
- *Неохідно підключити заголовочний файл*
  - #include "msp430.h"
  - #include "ioavr.h"

Приклад програми на C

```
#include "io430.h"
int main( void )
{
  //button
  P1DIR &= ~BIT3&0xff;
  P1REN |= BIT3;
  P1OUT |= BIT3;

  //led
  P1DIR |= BIT0;
  P1OUT &= ~BIT0&0xff;
  }
  while(1){
    if(P1IN&BIT3) continue;
    P1OUT^=BIT0;
    while(!(P1IN&BIT3));
  }
  return 0;
}
```

Асемблер VS C

- *Асемблер*
  - Максимально наближено до апаратного забезпечення
  - Малий код
  - Швидка програма
  - Складно розробляти
  - Погано переноситься

- Застосовується рідко
- *C*
  - Ближче до людини
  - Дещо більший код і повільніша програма
  - Просто розробляти код
  - Просто переносити програму для різних процесорів
  - Надзвичайно широко застосовується

Компіляція і компоновка

- *Перетворення коду програми у машинний код процесора*
- *Етапи*
  - Генерація об'єктного коду
  - Компоновка об'єкного коду в виконуваний файл
- *Об'єктний код*
  - Машинний код без адрес
  - Замість адрес інформація про положення (relocation)
  - Символи – імена змінних та функцій
  - Інформація для налагодження програм
- *Компоновка – зв'язування об'єктних файлів у виконуваний*
  - Присвоєння символам істинні значення адрес
  - Іноді адреси вказує користувач (для створення завантажувача)

Бібліотеки

- *Готові текстові чи об'єктні файли і файли описів*
- *Приклади бібліотек*
  - Математичні бібліотеки
  - Робота з алфавітно-цифровим індикатором
  - Вимірювання ємності
  - ..
- *Підключення бібілотеки*
  - Пдключення файла заголовку #include "HD44780LIB.h“
  - Компіляція додаткових тектових файлів (не завжди)
  - Компоновка проекта з додатковими об'єктними файлами

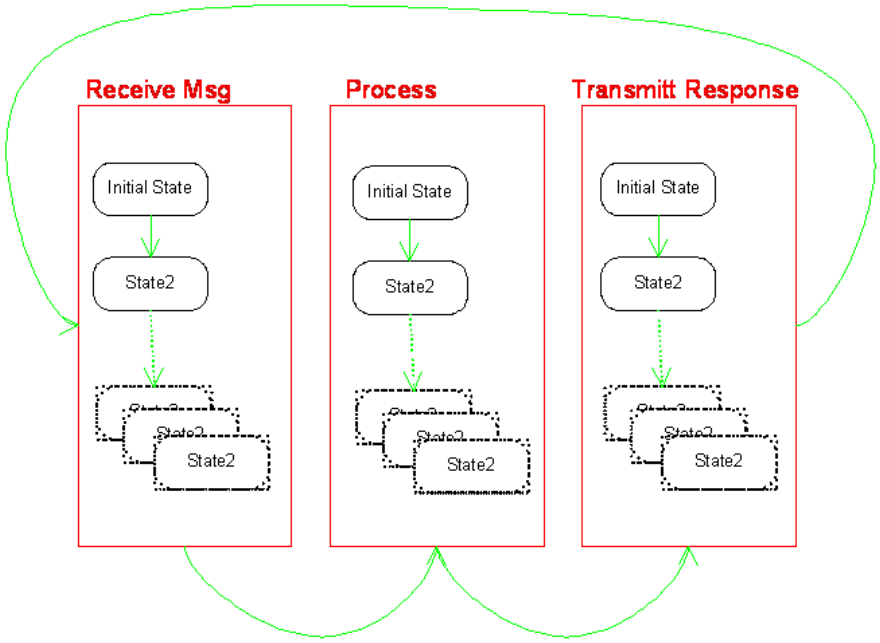
Операційні системи для мікроконтролерів

- *Програмне забезпечення для віртуалізації ресурсів мікроеконтролера*
  - Керування апаратним забезпеченням
  - Планування завдань
  - Однаковий інтерфейс для всіх завдань
  - “Така собі біліотека”
- *Для мікроконтролерів*
  - RTOS – робота в ральному часі
  - Мікро або екзоядро – надання функцій виділення ресурсів та/або перемикання між задачами
  - ОС компілюється разом з завдяннями і завантажується в мікроконтролер

Приклад операційої системи

```
void main ( void )
{
      initSystem();

while (1)
{
      work();
      sleep();
}
}
void work (void)
{
      doTask(RecieveMsg);
      doTask(Process);
      doTask(TransmittResponse);
}
__interrupt void Timer_A (void)
{
      wakeUp();
}
```



Прошивка мікроконтролера

- *Виконуваний файл треба завантажити в пам'ять мікроконтролера*
  - Спеціальний апаратний інтерфейс JTAG (Join Test and Access Group)
  - Спеціальна програма в постійній пам'яті мікроконтролера (Bootstrap loader)
  - Програма користувача у флеш пам'яті (loader)
- *Часто завантажувати можна прямо в схемі*

- ISP (In System Programming)
- Через SPI, I2C,UART переписувати ділянки пам’яті

Прошивка AVR через SPI

- *6 або 10 піновий роз’єм*
- *RESET керує вибором режиму прошивки, чи нормалної роботи*
- *Програматор є ведучим і генерує сигнал SCK*
- *Перехід в режим прошивки відбувається при певних співвідношеннях рівнів RESET та SCK*

Налагодження програм (debug)

- *Налагодження в симуляторі*
  - Програма компілюється із спеціальною інформацією наладки
  - Завантажується в програму наладки
  - Є можливість зупиняти програму в різних місцях
  - Дивитись вміст пам’яті та регістрів
  - Змінювати вмість пам’яті та регістрів
  - Дивитись стек викликів функцій
  - …
- *Налагодження в схемі*
  - Те ж саме тільки прямо в схемі – значно ефективніше
  - FET (flash emulation tool)
  - JTAG наладчики
  - Потрібні спеціальні пристрої (Launchpad підтримує)

Програматори

- *Ардуіно*
  - AVR
  - Не підтримує наладку в схемі

Симуляція

- *Моделювання роботи схеми в комп’ютерный програмі*
- *Симуляція коду*
  - TinkerCad
  - IAR Workbench
  - Proteus
  - VMLAB
- *Симуляція електричної схеми*
  - Proteus
  - PSpice
- *Повна симуляція*
  - Proteus
  - VMLAB

Середовища розробки програм

- *Спрощення редагування коду*
- *Спрощення компіляції*
- *Спрощення прошивки*
- *Спрощення симуляції*
- *Спрощення наладки*
- *Графічний інтерфейс користувача*
  - IAR Workbench (багато різних)
  - AVR studio (AVR
  - Arduino.cc (AVR)

Proteus (ISIS)

- *Створення схеми*
  - Додавання компонентів
  - Додавання з'єднань
  - Створити новий елемент
  - Розміщення елементів
- *Симуляція*
- *Наладка*
  - Двічі клацнути мишкою на елементі схеми

IAR Workbench

- *Створити WORKSPACE*
  - Вміст для проектів
  - MENU->New->Workspace
- *Створити проект*
  - Project->Create New Project
  - Одна програма
- *Редагуємо файли*

Опції для компіляції в IAR під FET і Proteus

- *В опціях проекта*

Вказати програми для мікроконтролера в Proteus

- *В параметрах мікросхеми*

Контрольні питання

1. Основні етапи програмування.

2. Асемблер vs C. Переваги та недоліки використання.

3. Переваги використання спеціалізованих середовищ розробки програм.

4. Як виконується програмування мікроконтролерів та налагодження програм.